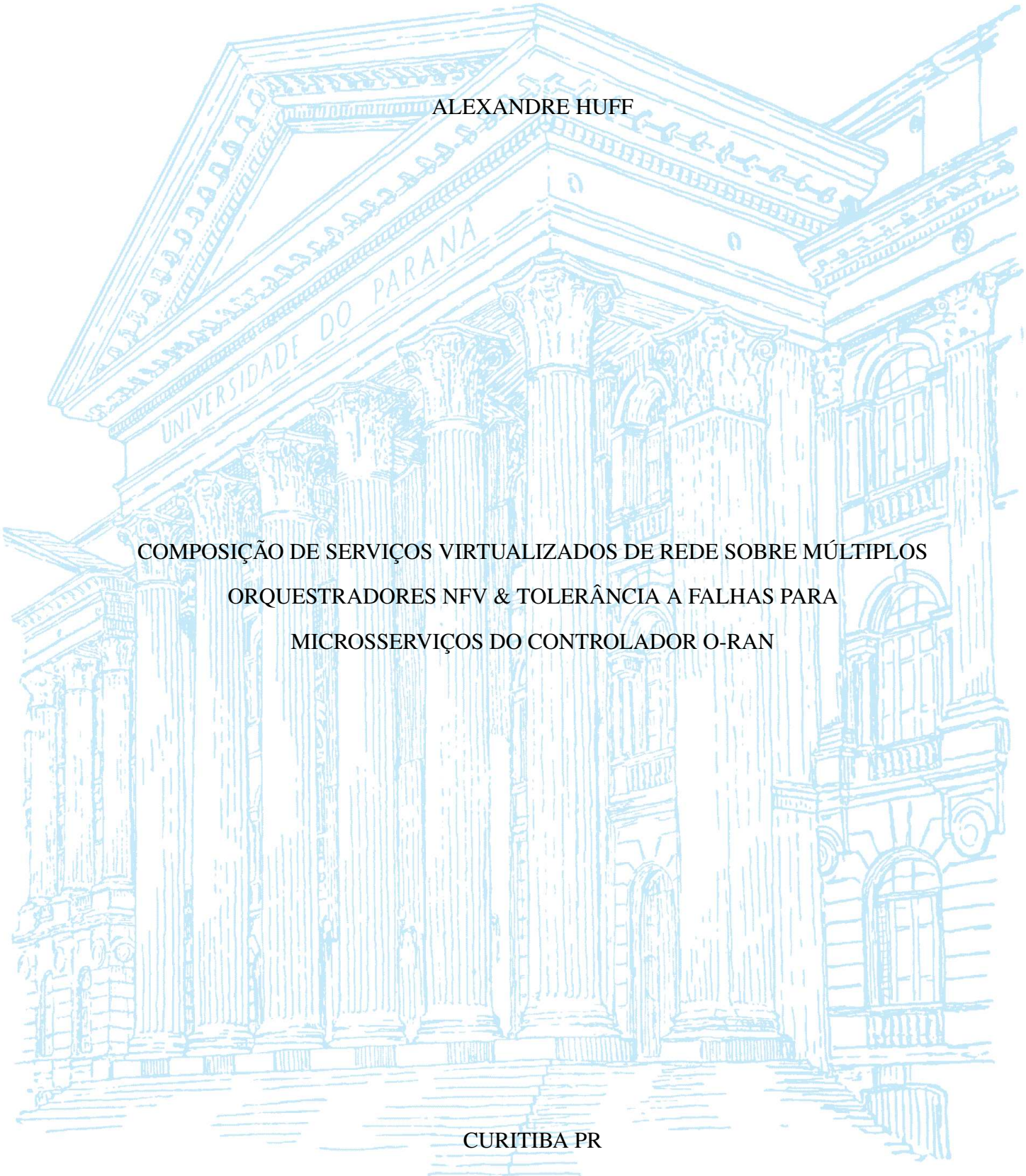
UNIVERSIDADE FEDERAL DO PARANÁ

ALEXANDRE HUFF



COMPOSIÇÃO DE SERVIÇOS VIRTUALIZADOS DE REDE SOBRE MÚLTIPLOS ORQUESTRADORES NFV & TOLERÂNCIA A FALHAS PARA MICROSSERVIÇOS DO CONTROLADOR O-RAN

CURITIBA PR

2021

ALEXANDRE HUFF

COMPOSIÇÃO DE SERVIÇOS VIRTUALIZADOS DE REDE SOBRE MÚLTIPLOS ORQUESTRADORES NFV & TOLERÂNCIA A FALHAS PARA MICROSSERVIÇOS DO CONTROLADOR O-RAN

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA PR

2021

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

H889c Huff, Alexandre
Composição de serviços virtualizados de rede sobre múltiplos orquestradores NFV & tolerância a falhas para microsserviços do controlador O-RAN [recurso eletrônico] / Alexandre Huff. – Curitiba, 2021.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2021.

Orientador: Elias Procópio Duarte Junior.

1. Telecomunicações - Equipamento e acessórios. 2. Recursos de rede de computador.
I. Universidade Federal do Paraná. II. Duarte Junior, Elias Procópio. III. Título.

CDD: 006.7876

Bibliotecária: Vanusa Maciel CRB- 9/1928

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **ALEXANDRE HUFF** intitulada: **COMPOSIÇÃO DE SERVIÇOS VIRTUALIZADOS DE REDE SOBRE MÚLTIPLOS ORQUESTRADORES NFV & TOLERÂNCIA A FALHAS PARA MICROSSERVIÇOS DO CONTROLADOR O-RAN**, sob orientação do Prof. Dr. ELIAS PROCÓPIO DUARTE JÚNIOR, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 11 de Fevereiro de 2021.

Assinatura Eletrônica
12/02/2021 14:36:42.0
ELIAS PROCÓPIO DUARTE JÚNIOR
Presidente da Banca Examinadora

Assinatura Eletrônica
12/02/2021 13:55:15.0
LUCIANO PASCHOAL GASPARY
Avaliador Externo (UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL)

Assinatura Eletrônica
19/02/2021 15:01:05.0
LUIS CARLOS ERPEN DE BONA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
12/02/2021 14:39:20.0
ROGÉRIO CORRÊA TURCHETTI
Avaliador Externo (UNIVERSIDADE FEDERAL DE SANTA MARIA)

Assinatura Eletrônica
12/02/2021 15:35:03.0
KLEBER VIEIRA CARDOSO
Avaliador Externo (UNIVERSIDADE FEDERAL DE GOIÁS)

AGRADECIMENTOS

Inicialmente gostaria de agradecer em especial à minha esposa, Léia Liliane, que esteve ao meu lado de forma incontestável por todo o período de estudos. Muito obrigado pela paciência, companheirismo, dedicação, palavras de motivação e por viabilizar o meu foco na pesquisa sem que eu tivesse preocupações adicionais. Eu não teria chegado tão longe sem a sua participação.

Gostaria de agradecer ao meu orientador, Prof. Elias P. Duarte Jr., pelo excelente direcionamento e ensinamentos durante todo o período do Doutorado. Muito obrigado pela disponibilidade para realizar discussões técnicas e filosóficas sobre os temas investigados nesta Tese, que inclusive ocorreram algumas vezes em finais de semana. Obrigado pelas palavras de motivação nos períodos difíceis da pesquisa. Muito obrigado por tornar possível o estágio sanduíche no exterior, o que contribuiu para um importante crescimento pessoal e profissional.

Agradeço aos professores do Programa de Pós-Graduação em Informática (PPGInf) pelos excelentes ensinamentos. Também agradeço aos meus colegas de laboratório (LARSIS) que participaram, de uma ou de outra forma, das discussões sobre o tema investigado e dos momentos de descontração, quando possível.

Gostaria de agradecer ao meu co-orientador do estágio sanduíche, Matti A. Hiltunen, pelos ensinamentos, discussões, disponibilidade e oportunidade de realizar pesquisa em uma área distinta e inovadora. Obrigado aos demais colegas de pesquisa da AT&T Shannon Labs pelas excelentes discussões. Muito obrigado à Dongmei Wang e Guangzhi Li pelo excelente acolhimento durante toda a minha estada no exterior. Agradeço ao Kermit Hal Purdy por sua ótima receptividade e auxílio.

I would like to thank my co-supervisor of the Doctoral Exchange Program at AT&T Shannon Labs, Matti A. Hiltunen, for his guidance, discussions, availability, and opportunity to conduct research in a distinct and innovative area. Thanks to the other research colleagues from the AT&T Shannon Labs for their outstanding discussions. Thanks to Dongmei Wang and Guangzhi Li for their excellent welcome throughout my stay abroad. I thank Kermit Hal Purdy for his great receptivity and assistance.

Agradeço aos meus pais e amigos por entenderem a minha ausência em determinados momentos e também pelas palavras de motivação sempre que nos reuníamos. Por fim, e não menos importante, o presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

A virtualização de funções de rede (*NFV - Network Function Virtualization*) permite a substituição de serviços disponibilizados tradicionalmente em hardware por software. Uma função virtualizada de rede (*VNF - Virtual Network Function*) é responsável por processar um determinado fluxo da rede e pode ser executada em hardware de prateleira. A ETSI (*European Telecommunications Standards Institute*) propôs a arquitetura NFV-MANO (*NFV - Management and Orchestration*) para o gerenciamento e orquestração da NFV. Uma SFC (*Service Function Chaining*) consiste no encaminhamento do fluxo da rede para um conjunto de VNFs em uma ordem pré-definida. No contexto da NFV, esta Tese de Doutorado apresenta o *Holistic-Composer* para a composição e o gerenciamento do ciclo de vida de uma SFC que pode ser orquestrada por diferentes plataformas NFV. Um *framework* foi definido e permite abstrair as configurações de baixo nível para compor e executar SFCs através de uma API genérica e única. Um protótipo foi implementado e experimentos são reportados avaliando a contribuição de forma quantitativa e qualitativa. Ainda no contexto da NFV, é proposta a Multi-SFC para a composição e execução de SFCs em múltiplas nuvens de uma federação de domínios orquestrados por diferentes plataformas NFV. A Multi-SFC utiliza uma abordagem holística e define um *framework* que fornece abstrações de alto nível para compor e executar as SFCs. Uma Multi-SFC é formada por segmentos que são interconectados por túneis implementados como VNFs. Um protótipo foi implementado como prova de conceito e resultados experimentais mostram que a Multi-SFC apresenta baixa latência e mantém vazão compatível com o túnel e hardware utilizados. Esta Tese de Doutorado também apresenta uma contribuição em outra área: as redes de acesso baseadas em rádio (*RAN - Radio Access Network*), que têm grande relevância no contexto de redes celulares 5G. Foi proposta uma estratégia de tolerância a falhas para o controlador RIC (*RAN Intelligent Controller*) definido na arquitetura O-RAN (*Open RAN*). A O-RAN visa padronizar elementos RAN, incentivando a interoperabilidade e implementações de código aberto. A estratégia de tolerância a falhas foi proposta tendo em vista os requisitos rígidos de latência e vazão da RAN. O RIC executa microsserviços diversos, denominados xApps, que permitem customizar o comportamento da RAN. A estratégia de tolerância a falhas proposta consiste de técnicas de particionamento de estado com replicação parcial em grupos de xApps e re-roteamento de mensagens com ciência de papel. Uma biblioteca chamada RFT (*RIC Fault Tolerance*) foi implementada e disponibilizada para o desenvolvimento de xApps tolerantes a falhas. Resultados experimentais mostram que os microsserviços tolerantes a falhas garantem os requisitos de baixa latência e alta vazão impostos pela RAN.

Palavras-chave: NFV. SFC. Composição. Orquestração. Tolerância a Falhas. O-RAN.

ABSTRACT

Network Function Virtualization (NFV) allows the implementation in software of network services that are traditionally available as middleboxes deployed on specialized hardware. A Virtual Network Function (VNF) can be executed on commercial off-the-shelf hardware. The European Telecommunications Standards Institute (ETSI) has proposed the NFV Management & Orchestration (NFV-MANO) architecture with the purpose of allowing NFV interoperability. While a VNF processes a network flow, an SFC (Service Function Chaining) consists of steering the network flow along a sequence of VNFs in a predefined order. SFCs allow the composition of VNFs to enable the network to offer complex services. The first contribution of this Thesis is the the Holistic-Composer, which allows the composition and life cycle management of an SFC that can be orchestrated by different NFV platforms. A framework was defined which abstracts the minute low-level details for the composition and management of SFCs through a single and generic API. We implemented a prototype as a proof of concept, and experiments were executed to evaluate the solution quantitatively and qualitatively. The second contribution of the Thesis is the Multi-SFC, which allows the composition and execution of SFCs across federated clouds and domains orchestrated by different NFV platforms. The Multi-SFC employs a holistic approach and defines a framework that provides high-level abstractions to compose and execute SFCs. A Multi-SFC consists of segments that are interconnected by tunnels implemented as VNFs. We implemented a Multi-SFC prototype as a proof of concept, and experimental results show that the Multi-SFC presents low latency and high throughput which vary depending on the tunnel technology and hardware employed. In addition to the NFV, this Thesis also presents a contribution in another area: Radio Access Network (RAN) which has become highly relevant in the context of 5G cellular networks. We propose a fault tolerance strategy to the RAN Intelligent Controller (RIC) defined in the O-RAN (Open RAN) architecture. O-RAN aims to standardize RAN elements, encouraging interoperability and open source implementations. The RIC is a platform for implementing RAN control functions as microservices called xApps, which effectively allows a RAN to be customized. The proposed strategy takes into account the strict RAN requirements in terms of latency and throughput. We propose techniques that use state partitioning, partial replication, and fast re-route with role awareness to decrease the overhead. We implemented the fault tolerance techniques as a library, called RFT (RIC Fault Tolerance), that xApp developers can employ to implement fault-tolerant xApps. Performance results show that RFT meets the latency and throughput requirements as the number of xApp replicas increases.

Keywords: NFV. SFC. Composition. Orchestration. Fault Tolerance. O-RAN.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

3GPP	3rd Generation Partnership Project
5G	Fifth Generation of Cellular Network Technology
ACK	Acknowledgment
ACL	Access Control List
API	Application Programming Interface
ASN.1	Abstract Syntax Notation 1
AT&T	American Telephone & Telegraph
BGP	Border Gateway Protocol
CAPEX	CApital EXpenditures
CLI	Command Line Interface
CP	Connection Point
CUSTOM	CUstom Service TOpology Model
DDoS	Distributed Denial of Service
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
EM	Element Management
ETSI	European Telecommunications Standards Institute
FCAPS	Fault, Configuration, Accounting, Performance and Security
FENDE	Federated Ecosystem for OfferiNg, Distribution, and Execution of Virtual Network Functions
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
IETF SFC WG	IETF Service Function Chaining Working Group
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IdP	Identity Provider
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
LTE	Long Term Evolution
MDSO	Multi-Domain Service Orchestration
MPLS	Multiprotocol Label Switching

NACK	Negative-acknowledgment
NAT	Network Address Translation
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFV ISG	NFV Industry Specification Group
NFV-MANO	NFV - Management and Orchestration
NFVO	NFV Orchestrator
NS	Network Service
NSH	Network Service Header
OAI	Open Air Interface
ODL	OpenDaylight
ONAP	Open Network Automation Platform
OPEX	OPerational EXpenditures
OPNFV	Open Platform for NFV
O-RAN	Open RAN
O-RAN SC	O-RAN Software Community
OSS/BSS	Operations Support System / Business Support System
OSM	Open Source MANO
QoS	Quality of Service
RAN	Radio Access Network
RBAC	Role-based Access Control
RDCL 3D	Reusable Functional Blocks Description and Composition Language Design, Deploy and Direct
REST	REpresentational State Transfer
RFT	RIC Fault Tolerance
RIC	RAN Intelligent Controller
RMR	RIC Message Router
SAML	Security Assertion Markup Language
SCTP	Stream Control Transmission Protocol
SDK	Software Development Kit
SDL	Shared Data Layer
SDN	Software-Defined Networking
SFC	Service Function Chaining
SFCs	Service Function Chains
SFF	Service Function Forwarder
SFP	Service Function Path
SP	Service Provider
SRv6	Segment Routing v6

TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
USA	United States of America
VIM	Virtualized Infrastructure Manager
VNF	Virtual Network Function
VNFD	VNF Descriptor
VNFFG	VNF Forwarding Graph
VNFFGD	VNF Forwarding Graph Descriptor
VNFM	VNF Manager
VPN	Virtual Private Network
VXLAN	Virtual eXtensible Local Area Network
YANG	Yet Another Next Generator
YAML	Ain't Markup Language
WSGI	Web Server Gateway Interface

SUMÁRIO

1 INTRODUÇÃO

A Virtualização de Funções de Rede (NFV - *Network Function Virtualization*) permite a implementação em software de diversas funções das redes que tradicionalmente são executadas em *middleboxes*. Uma função virtualizada de rede é denominada VNF (*Virtual Network Function*) e pode ser executada por sistemas de virtualização em hardware de prateleira, aumentando a flexibilidade e diminuindo custos (Chiosi et al., 2013; Martins et al., 2014; Yousaf et al., 2017; Yi et al., 2018; de Sousa et al., 2019). Cada VNF é responsável pelo processamento específico de determinado fluxo da rede e pode operar em diferentes camadas da pilha de protocolos. Nesse contexto, a ETSI (*European Telecommunications Standards Institute*) propôs a arquitetura NFV-MANO (NFV - *Management and Orchestration*) (Quittek et al., 2014) para o gerenciamento e orquestração de funções virtualizadas de rede.

Um dos principais objetivos do NFV-MANO é coordenar a composição de VNFs para formar SFCs (*Service Function Chains*) que permitem prover serviços complexos de rede. Uma SFC consiste de uma composição de VNFs em uma topologia virtual pela qual o encaminhamento do tráfego ocorre em uma ordem pré-definida (Halpern e Pignataro, 2015; Garcia et al., 2020). O tráfego da rede de uma SFC normalmente é encaminhado entre as VNFs com base em um identificador de fluxo, enquanto que no roteamento convencional as decisões de encaminhamento geralmente são baseadas no endereço IP de destino. A expressão *service chain* também é utilizada para referir-se a uma SFC, ambas são utilizadas como sinônimos no texto.

Esta Tese de Doutorado apresenta contribuições em dois contextos diferentes: a composição e execução de serviços de rede com múltiplos domínios e orquestradores NFV; e, a tolerância a falhas para microsserviços do controlador O-RAN que é introduzida mais à frente. A primeira contribuição deste trabalho no contexto da NFV é o *Holistic-Composer* (Huff et al., 2018b,a), que permite a composição e o gerenciamento do ciclo de vida de uma SFC em diferentes orquestradores NFV. Na abordagem proposta, o gerenciamento do ciclo de vida da SFC consiste de um conjunto de funções que permitem, por exemplo, a instanciação, o monitoramento, o encerramento e a alocação dos recursos necessários para a sua execução. O *Holistic-Composer* foi definido como uma solução genérica e extensível baseada nos padrões NFV-MANO especificados pela ETSI. A segunda contribuição também no contexto da NFV é a Multi-SFC (Huff et al., 2019; Huff et al., 2020). A Multi-SFC é uma abordagem que permite a composição e execução de SFCs que atravessam múltiplas nuvens em domínios federados e orquestrados por diferentes plataformas NFV.

A terceira contribuição da Tese vem em outro contexto, que surgiu durante o estágio sanduíche na AT&T Shannon Labs - Research. O objetivo inicial do estágio sanduíche era trabalhar com NFV no contexto da plataforma ONAP (2020), da qual a AT&T é uma das principais mantenedoras. Entretanto, por motivos diversos, optou-se por não seguir aquela linha e trabalhar em um projeto distinto que tem recebido grande atenção da comunidade internacional, a O-RAN (*Open RAN*) (O-RAN Alliance, 2019). A Rede de Acesso por Rádio (RAN - *Radio Access Network*) têm promovido uma verdadeira revolução na forma com que pessoas e máquinas comunicam e interagem (Olwal et al., 2016; Parvez et al., 2018; Habibi et al., 2019). A tecnologia 5G (Shayea et al., 2020) corresponde a mais recente geração da RAN e visa fornecer serviços móveis ubíquos com alta qualidade (QoS - *Quality of Service*). A tecnologia promete impactar as mais variadas áreas de aplicação como cidades inteligentes, veículos autônomos, assistência médica, segurança pública, entretenimento, realidade aumentada e automação industrial e residencial. Vale destacar que a IoT (*Internet of Things*) (Dorsemaine

et al., 2015) viabiliza diversas dessas áreas de aplicação e é viabilizada pela RAN (Zayas e Merino, 2017). A IoT deverá gerar um extraordinário crescimento no volume de tráfego e na densidade de equipamentos conectados à RAN (Habibi et al., 2019; 3GPP, 2020b). Há previsões (GSMA, 2020; 3GPP, 2020b) de crescimento entre 2020 e 2025 de 3,8 para 5 bilhões de usuários que se conectam à Internet em dispositivos móveis; e de 12 para 24,6 bilhões de dispositivos IoT conectados à RAN. Além disso, a densidade de conexões para a RAN chega a projeções de 1.000.000 de equipamentos por KM^2 até 2025.

Recentemente, a O-RAN Alliance (2019) e a O-RAN SC (2020b) (*O-RAN Software Community*) vêm liderando esforços para apoiar o desenvolvimento de software de código aberto para a RAN, considerando diversos requisitos de desempenho, latência, escalabilidade e alinhamento com os padrões especificados pelo 3GPP (*3rd Generation Partnership Project*) (3GPP, 2020a). A O-RAN SC (2020b) também vem definindo uma interface de comunicação para os elementos da RAN visando permitir que equipamentos proprietários sejam capazes de expor suas funcionalidades e reportar notificações de forma padronizada. Além disso, a O-RAN SC (2020b) vêm liderando esforços para a implementação e disponibilização de um controlador O-RAN de código aberto denominado RIC (*RAN Intelligent Controller*) (O-RAN SC, 2020a). O objetivo do RIC é fornecer uma plataforma extensível para executar funções de controle e customizar o comportamento da RAN. Funções de controle diversas são implementadas e adicionadas à plataforma RIC como microsserviços chamados xApps. A interação entre os xApps e os elementos da RAN apresenta requisitos rígidos de latência e escalabilidade (Coletti et al., 2018; Akman et al., 2020; O-RAN Alliance, 2020c,a).

A terceira contribuição desta Tese é uma estratégia para a tolerância a falhas de xApps e portanto do próprio controlador RIC. A abordagem para implementar a tolerância a falhas é baseada na replicação de xApps. São utilizadas técnicas de particionamento de estado com replicação parcial e re-roteamento com ciência de papel para garantir a disponibilidade e, ao mesmo tempo, atender aos requisitos de desempenho impostos pela RAN. Uma biblioteca denominada RFT (*RIC Fault Tolerance*) foi implementada e disponibilizada para o desenvolvimento de xApps tolerantes a falhas.

Nas seções seguintes são detalhados os problemas de pesquisa, as hipóteses do trabalho e as contribuições da Tese, finalizando com uma descrição da organização do texto.

1.1 PROBLEMAS

A segmentação do mercado gera uma grande quantidade de redes de telecomunicações e operadoras de nuvem que disponibilizam seus serviços em regiões específicas. Isso resulta em tarefas desafiadoras para implantar infraestruturas de serviços de baixo custo (*e.g.*, serviços de computação e de conectividade de rede) que podem inclusive abranger vários países. Mesmo as grandes operadoras de telecomunicações e de nuvem não possuem abrangência global de infraestrutura (Bernardos et al., 2019). Provedores de serviços geralmente utilizam interfaces de comunicação exclusivas e proprietárias que resultam em tarefas complexas e lentas para integrar serviços de rede em múltiplos domínios com base na composição de SFCs (de Sousa et al., 2019; Cui et al., 2020). Essa abordagem de integração é ineficiente e cara pois requer esforços significativos de especialistas de rede em cada domínio – idealmente estes especialistas deveriam concentrar o foco apenas nas regras de negócio durante a composição de um novo serviço.

Embora as operadoras de nuvem e de telecomunicações forneçam serviços de infraestrutura virtualizados (*e.g.*, rede, computação e armazenamento), ainda há uma necessidade de maior integração entre as mesmas para oferecer suporte à composição e orquestração de serviços de rede que atravessam múltiplas nuvens/domínios/plataformas. As plataformas NFV atuais são

limitadas e complexas no aspecto de colaboração entre orquestradores NFV para a composição e execução de serviços de rede inter-domínios (Bernardos et al., 2019; Cui et al., 2020).

As plataformas NFV existentes geralmente permitem que a instanciação e orquestração das VNFs de uma SFC sejam realizadas em uma mesma plataforma NFV (Tacker, 2020; ETSI, 2020b; Ciena, 2020; Cloudify, 2020; ONAP, 2020; Sonkoly et al., 2015). Embora algumas plataformas suportem múltiplas instâncias de um mesmo orquestrador NFV, desconhece-se plataformas NFV que suportem a composição e o gerenciamento do ciclo de vida de uma SFC em domínios federados que são orquestrados por plataformas NFV de fabricantes distintos. Conforme múltiplas e diferentes plataformas NFV se tornam disponíveis (Ghaznavi et al., 2017; de Sousa et al., 2019), é mais do que natural que uma SFC seja composta e orquestrada sobre plataformas NFV distintas.

Embora a ETSI esteja definindo interfaces para a comunicação entre orquestradores NFV de diferentes domínios administrativos, o problema ainda não foi resolvido (Haitao e Mann, 2018; Haitao et al., 2018; Yousaf et al., 2020). Operadores de rede ainda necessitam lidar com as particularidades de operação e configuração de orquestradores NFV distintos que também implementam diferentes modelos de dados. A ETSI vem discutindo a composição e execução de serviços de rede em diferentes domínios administrativos através do encadeamento de SFCs existentes (*i.e.*, SFCs compostas previamente são aninhadas para formar novas SFCs), mas não considera a possibilidade de compor novas SFCs a partir de VNFs independentes de serviços de rede previamente estabelecidos. Destaca-se que esta possibilidade de compor novas SFCs a partir de VNFs independentes é muito natural, trazendo grande flexibilidade para a disponibilização de novos serviços sobre redes federadas. Além disso, a estratégia prevista pela ETSI de aninhar composições de SFCs para criar novos serviços pode gerar desperdício de recursos computacionais e energia, uma vez que o tráfego da rede pode ter que atravessar VNFs desnecessárias.

A composição de SFCs com VNFs independentes e que executam em diferentes domínios administrativos pode ser implementada com o apoio de federações. Uma federação de domínios possibilita que VNFs sejam instanciadas em diferentes domínios com plataformas NFV distintas para formar serviços complexos de rede. As instâncias de VNFs em execução também podem ser compartilhadas por múltiplas SFCs caso o acordo de federação autorizar e as plataformas e orquestradores NFV suportarem essa funcionalidade.

Mudando para a outra área na qual a Tese apresenta uma contribuição, a RAN está evoluindo como parte de um conjunto de novas tecnologias (incluindo 5G, IoT, entre outras) que tem potencial de grande impacto em diversos contextos. Uma RAN tem a tarefa não trivial de determinar a melhor forma de utilização e gerenciamento do espectro limitado da rede sem fio para possibilitar a conectividade de equipamentos de usuário (UE - *User Equipment*). Em redes 5G densas (*i.e.*, muitas estações base e UEs em uma área geográfica limitada) torna-se ainda mais desafiador alocar recursos de rádio, implementar *handovers* (*i.e.*, transferir a conexão de um UE de uma estação base para outra), gerenciar interferências, balancear a carga entre células da rede, entre outras tarefas que a RAN deve executar. Uma célula consiste em uma determinada área geográfica em que há cobertura de sinal da RAN (Gudipati et al., 2013; Habibi et al., 2019).

Embora as interfaces de comunicação entre os dispositivos sem fio e os elementos da RAN sejam definidas por padrões abertos, a maioria das implementações são normalmente soluções fechadas, proprietárias, desenvolvidas pelos fabricantes de equipamentos (Olwal et al., 2016; Parvez et al., 2018; Habibi et al., 2019). Nesse contexto, provedores de telecomunicação enfrentam diversos desafios para realizar aprimoramentos bem como desenvolver novos serviços e, ao mesmo tempo, torna-se complexo e lento para a comunidade de pesquisa contribuir com essa importante área das redes de telecomunicações.

Projetos como o srsLTE (Gomez-Miguelez et al., 2016) e o OAI (*Open Air Interface*) (Kaltenberger et al., 2019) iniciaram implementações de código aberto para a RAN considerando os padrões do 3GPP (3GPP, 2020a). Recentemente a O-RAN Alliance (2019) e a O-RAN SC (2020b) vêm definindo uma interface de comunicação entre os elementos da RAN e implementado um controlador para a RAN denominado RIC. As funções de controle da RAN são implementadas por meio de microsserviços chamados xApps. Os xApps interagem com os elementos da RAN com requisitos rígidos de latência ao passo que devem prover escalabilidade para processar dezenas de milhares de requisições por segundo em uma instância do RIC (Coletti et al., 2018; Akman et al., 2020; O-RAN Alliance, 2020c,a).

A alta disponibilidade é considerada requisito fundamental para a RAN (Habibi et al., 2019). Um xApp falho pode impedir que milhares de dispositivos sem fio conectem-se à rede em uma determinada área geográfica. Além de alta disponibilidade, as redes 5G demandam requisitos rígidos de baixa latência e alta vazão para as funções de controle da RAN (Parvez et al., 2018; Habibi et al., 2019). A alta disponibilidade pode ser alcançada por meio de técnicas de replicação e tolerância a falhas. Entretanto, as estratégias empregadas com as técnicas de replicação tradicionais para fornecer tolerância a falhas, descritas na Seção 5.8, não são capazes de suportar os requisitos de baixa latência e alta vazão exigidos pelas redes 5G no contexto do controlador RIC (O-RAN Alliance, 2020c,a). A plataforma RIC está sendo desenvolvida sob a responsabilidade da O-RAN SC (2020b) e, até onde se sabe, não existem propostas para tolerância a falhas de xApps e do controlador RIC.

1.2 HIPÓTESES

Esta Tese de Doutorado levanta duas hipóteses fundamentais. A primeira hipótese está relacionada ao contexto da composição e execução de serviços de rede com múltiplos domínios e orquestradores NFV. A segunda hipótese está associada ao contexto de tolerância a falhas para o controlador O-RAN.

A primeira hipótese levantada para conduzir a pesquisa deste Trabalho de Doutorado é a seguinte:

Hipótese 1: *Para aproveitar ao máximo a segmentação do mercado de telecomunicações e a grande quantidade de plataformas e orquestradores NFV disponíveis, estratégias efetivas e de alto nível devem permitir a composição e execução de serviços complexos de rede que atravessam múltiplos domínios orquestrados por diferentes plataformas NFV.*

Embora as estratégias atuais considerem a segmentação do mercado de telecomunicações e a disponibilização de diferentes plataformas de nuvem, falham ao desconsiderar que SFCs podem ser construídas sobre múltiplos orquestradores NFV distintos em uma federação. Ainda que a ETSI tenha discutido a padronização de interfaces para a comunicação entre orquestradores NFV e também a composição de SFCs multi-domínio, o foco é no encadeamento de serviços de rede existentes e disponibilizados previamente pelas interfaces dos orquestradores NFV. Isto pode restringir, por exemplo, a composição e a implantação de serviços de rede entre domínios federados que possuem suas próprias implementações de VNFs e apenas têm a necessidade de locar os serviços da plataforma subjacente de orquestração. Nesse sentido, estratégias mais flexíveis são necessárias para permitir a escolha do domínio mais adequado que um serviço de rede – ou parte dele – deve executar (*e.g.*, custo, desempenho, localização). Possibilitar a execução de um serviço de rede composto por VNFs de múltiplos e diferentes orquestradores NFV também traz benefícios em outros contextos na NFV, por exemplo permitindo aumentar

a flexibilidade de algoritmos de posicionamento de VNFs e SFCs, que podem aproveitar os múltiplos diferentes domínios/orquestradores da federação.

Um dos objetivos desta Tese de Doutorado é confirmar a **Hipótese 1** descrita acima e responder as seguintes questões de pesquisa:

1. Uma vez que existe uma grande quantidade de plataformas e orquestradores NFV, é possível abstrair as especificidades dos mesmos para compor e implantar SFCs através de uma API genérica e única?

2. Ao abstrair as especificidades das plataformas e orquestradores NFV, é possível compor e executar uma SFC em múltiplos domínios orquestrados por diferentes plataformas NFV sem a intervenção manual dos operadores de rede de cada domínio?

3. Considerando uma federação de domínios, um usuário com identidade federada é capaz de compor e executar uma SFC de forma flexível utilizando VNFs independentes dos serviços de rede providos pelas interfaces dos orquestradores NFV de cada domínio?

A segunda hipótese levantada para conduzir a pesquisa deste Trabalho de Doutorado no contexto da O-RAN é a seguinte:

Hipótese 2: *Para construir um controlador RIC tolerante a falhas e ao mesmo tempo atender aos requisitos de baixa latência e alta vazão, podem ser utilizadas técnicas de replicação e particionamento de estados, aliadas a estratégias eficientes de roteamento de mensagens da RAN para as réplicas apropriadas.*

Ao longo dos anos, diversas técnicas de tolerância a falhas baseadas em replicação de estado foram propostas. Entretanto, a forma como as técnicas de replicação tradicionais são empregadas para prover tolerância a falhas não suportam os requisitos de baixa latência e alta vazão requeridas no contexto do controlador RIC da O-RAN.

Esta Tese de Doutorado também objetiva confirmar a **Hipótese 2** e responder a seguinte questão de pesquisa:

4. É possível construir xApps tolerantes a falhas que são capazes de processar mensagens da RAN com a baixa latência e alta vazão requisitadas?

1.3 CONTRIBUIÇÕES

Este Trabalho de Doutorado apresenta contribuições em dois contextos diferentes de pesquisa: a composição e execução de serviços de rede com múltiplos domínios e orquestradores NFV; e, a tolerância a falhas para microsserviços do controlador O-RAN.

A primeira contribuição deste Trabalho de Doutorado é uma abordagem que permite a composição e o gerenciamento do ciclo de vida de uma SFC que pode ser orquestrada por diferentes plataformas NFV. No contexto da abordagem proposta, o gerenciamento do ciclo de vida representa um conjunto de funções que permitem alocar recursos de hardware, instanciar, monitorar e encerrar uma SFC (ETSI, 2014b). Um *framework* denominado *Holistic-Composer* foi definido e está centrado como uma solução genérica e extensível empregando os padrões da especificação NFV-MANO da ETSI. O *framework* é denominado holístico, pois o objetivo principal é abstrair as especificidades de diferentes orquestradores NFV para a composição de uma SFC através de uma API genérica e única. A abordagem proposta para o *Holistic-Composer*

emprega agentes de comunicação que traduzem as operações genéricas desta API para as operações específicas dos respectivos orquestradores NFV.

Um protótipo do *Holistic-Composer* foi implementado e permite realizar a composição e o gerenciamento do ciclo de vida das SFCs formadas por VNFs que executam o *unikernel* Click-on-OSv (da Cruz Marcuzzo et al., 2017) sobre a plataforma de orquestração NFV OpenStack Tacker (Tacker, 2020). Vale ressaltar que a composição de SFCs formadas por VNFs que executam outras tecnologias e sistemas operacionais também é permitida. Experimentos foram executados e permitiram realizar uma avaliação quantitativa e qualitativa. Na avaliação quantitativa é possível observar que são necessários em média 17,5ms para adicionar novas VNFs na SFC por um único cliente. A composição de 128 SFCs de forma concorrente foi 4,145 vezes mais rápida se comparada com a abordagem sequencial. Na avaliação qualitativa é possível verificar que o *Holistic-Composer* é capaz de compor uma SFC apenas com a informação da sequência de VNFs da composição.

A segunda contribuição da Tese é a Multi-SFC: uma abordagem que permite a composição e execução de uma SFC em múltiplas nuvens e domínios orquestrados por diferentes plataformas NFV em uma federação. A Multi-SFC utiliza uma abordagem holística e define um *framework* que fornece abstrações de alto nível para compor e executar as SFCs distribuídas nas múltiplas plataformas NFV. O bloco de construção básico da Multi-SFC é chamado de *segmento*, no qual todas as VNFs estão conectadas em uma determinada nuvem/domínio/orquestrador. Segmentos são interconectados por meio de túneis baseados em diferentes tecnologias como VPN (*Virtual Private Network*) e VXLAN (*Virtual eXtensible LAN*) que são instanciados como VNFs nos pontos de entrada e saída dos segmentos SFC sendo conectados. A configuração da infraestrutura de rede virtual para a interconexão dos túneis também é abstraída pela Multi-SFC. Foi ainda realizado um levantamento do conjunto de operações que diferentes orquestradores NFV implementam para compor suas SFCs. Com base nesse levantamento foi definida uma API genérica que permite abstrair as particularidades de diferentes plataformas NFV para compor e executar uma SFC que atravessa uma federação de domínios.

Um protótipo da Multi-SFC foi implementado com base nas plataformas NFV Tacker (Tacker, 2020) e Open Source MANO (ETSI, 2020b), e, com duas versões diferentes do OpenStack (OpenStack, 2020a) em dois domínios distintos. Resultados experimentais avaliam a Multi-SFC em termos de interoperabilidade e desempenho da rede, em especial a vazão e latência. Os resultados mostram que a abordagem da Multi-SFC apresenta baixa latência e mantém vazão compatível com o túnel e hardware utilizados na execução. A sobrecarga apresentada pelos túneis VNF está relacionada ao protocolo de comunicação utilizado e varia conforme o tipo de serviço que cada um deles é capaz de prover (*e.g.*, criptografia).

A terceira contribuição está relacionada ao contexto de tolerância a falhas para microsserviços do controlador O-RAN. A contribuição consiste na definição de um conjunto de técnicas para implementar xApps tolerantes a falhas capazes de suportar os requisitos de baixa latência e alta vazão na plataforma RIC desenvolvida pela O-RAN SC (2020b). A abordagem necessária para implementar a tolerância a falhas no controlador RIC é a replicação de xApps. A proposta é a utilização de técnicas de particionamento de estado com replicação parcial e re-roteamento com ciência de papel para atender aos requisitos de latência e vazão impostos pela RAN. O particionamento de estado de um xApp permite que um subconjunto de dados seja mantido por diferentes réplicas daquele xApp, enquanto que a replicação parcial permite que cópias de cada subconjunto de dados sejam mantidas por determinadas réplicas de *backup*. A ciência de papel possibilita que uma réplica reconheça o papel (*i.e.*, primário ou *backup*) que exerce sobre um determinado subconjunto de dados, enquanto que o re-roteamento garante que uma mensagem

seja encaminhada para uma réplica de *backup* caso a réplica primária não esteja disponível para processar uma mensagem da RAN.

As técnicas investigadas foram implementadas e disponibilizadas em uma biblioteca chamada RFT (*RIC Fault Tolerance*) (Huff et al., 2020). A RFT visa permitir que desenvolvedores de aplicações de controle da RAN implementem xApps tolerantes a falhas. Resultados experimentais mostram que a RFT é capaz de sustentar os requisitos de latência do RIC e também fornece uma taxa de transferência escalável de centenas de milhares de requisições por segundo.

1.4 ORGANIZAÇÃO DO TRABALHO

O restante do trabalho está organizado da seguinte forma. O Capítulo 2 descreve os fundamentos relacionados à NFV, os conceitos de orquestração de funções virtualizadas de rede e de *service chains*. No Capítulo 3, são apresentadas a arquitetura, implementação e avaliações do *framework Holistic-Composer*. Em seguida, o Capítulo 4 detalha a proposta da Multi-SFC que provê abstrações para realizar a composição e execução de SFCs distribuídas em múltiplas nuvens de domínios federados e orquestrados por diferentes plataformas NFV. A proposta de implementação de xApps tolerantes a falhas no controlador O-RAN é descrita no Capítulo 5. As conclusões finais e os trabalhos futuros são descritos no Capítulo 6.

2 VIRTUALIZAÇÃO DAS FUNÇÕES DE REDE

Este capítulo apresenta o paradigma NFV de virtualização de funções de rede. Inicialmente são descritas as funções virtualizadas de rede. Em seguida, são apresentados os conceitos e a arquitetura para o gerenciamento e orquestração de serviços de rede. Por fim, é detalhada a conceitualização e os componentes lógicos que habilitam a execução de uma SFC.

2.1 FUNÇÕES VIRTUALIZADAS DE REDE

Atualmente as redes de computadores disponibilizam funcionalidades diversas implementadas por uma variedade grande de dispositivos com finalidades específicas conhecidos como *middleboxes* (Martins et al., 2014). Esses dispositivos geralmente combinam hardware e software de um mesmo fabricante e oferecem funcionalidades importantes como, *firewalls*, *proxies*, balanceadores de carga e otimizadores de tráfego na Internet. *Middleboxes* têm, em geral, custo elevado de aquisição e operação, além de oferecerem dificuldades em termos de escalabilidade conforme aumenta a demanda da rede (Martins et al., 2014).

A Virtualização das Funções de Rede (NFV - *Network Function Virtualization*) surge como uma alternativa para substituir os *middleboxes*, permitindo que as funcionalidades tradicionais da rede sejam implementadas em software, e disponibilizadas por sistemas virtualização executados em hardware de prateleira (Chiosi et al., 2013; Han et al., 2015). Uma Função Virtualizada de Rede (VNF - *Virtual Network Function*) pode realizar tarefas de equipamentos específicos tais como, dispositivos NAT (*Network Address Translation*), roteadores, balanceadores de carga, *firewalls*, entre muitos outros (Han et al., 2015).

A Figura 2.1 ilustra a abordagem da NFV, na qual é possível visualizar que as funções de rede normalmente disponibilizadas por fabricantes de hardware específicos (*Network Vendor-Specific Hardware*) podem ser implementadas em software, e executadas em hardware de prateleira (*Commercial off-the-shelf*). Além disso, diversos benefícios podem ser alcançados com a implantação da NFV, dentre eles pode-se destacar: redução de custos com equipamentos e infraestrutura (CAPEX - *CAPital EXpenditures*); redução de custos com operacionalização dos equipamentos (OPEX - *OPerational EXpenditures*); desacoplamento de hardware e software; redução na variedade de equipamentos; maior flexibilidade no gerenciamento e orquestração de funções de rede; recuperação ágil de falhas; agilidade na entrega de funções de rede; e, alterações na topologia da rede em tempo de execução (Chiosi et al., 2012).

Os paradigmas das Redes Definidas por Software (SDN - *Software-Defined Networking*) (Kim e Feamster, 2013) e da Computação em Nuvem (*Cloud Computing*) (Zhang et al., 2010) complementam a tecnologia NFV. As redes SDN permitem desacoplar o plano de controle da rede do plano de dados que somente encaminha pacotes. As funções do plano de controle são executadas por um controlador externo e submetidas ao dispositivo de encaminhamento por meio de protocolos padronizados (Trois et al., 2016). Já o modelo da computação em nuvem, permite o acesso sob demanda e pela rede a recursos computacionais compartilhados que podem ser rapidamente disponibilizados e liberados com o mínimo esforço de gerenciamento ou interação do provedor de serviços (Zhang et al., 2010; Mell e Grance, 2011).

Ainda que os mecanismos para o desenvolvimento das funções virtualizadas de rede não dependam da SDN, a separação do plano de controle do plano de dados pode simplificar a implantação da NFV com serviços existentes, melhorar o desempenho e facilitar a operação da rede. Por outro lado, o modelo da SDN pode usufruir da NFV, pois esta é capaz de oferecer a

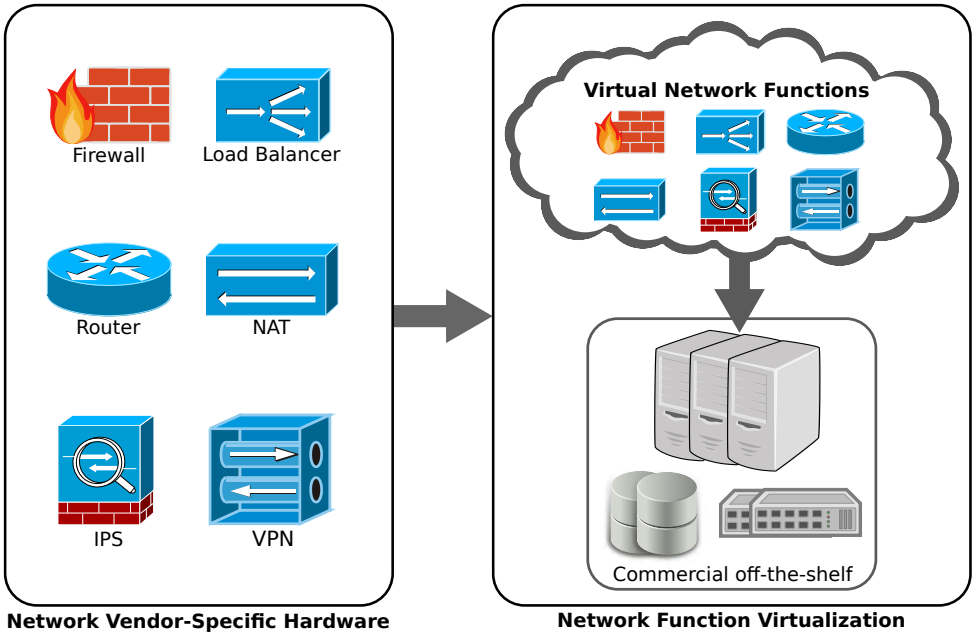


Figura 2.1: Abordagem da NFV (Chiosi et al., 2012).

infraestrutura necessária para as aplicações SDN executarem (Chiosi et al., 2012). A infraestrutura da NFV também pode aproveitar-se das capacidades da computação em nuvem e da virtualização permitindo que funcionalidades como *firewalls*, balanceamento de carga, inspeção de tráfego, entre outras, sejam executadas em software e gerenciadas por um mecanismo de orquestração (Mijumbi et al., 2016; de Sousa et al., 2019).

2.2 ORQUESTRAÇÃO DE FUNÇÕES VIRTUALIZADAS DE REDE

A gerência e operação das redes tem se tornado mais complexas e, portanto, mais sujeitas a erros devido à variedade de operações e *scripts* que devem ser executados. Como exemplo, pode-se citar o escalonamento e compartilhamento de recursos, gerenciamento de falhas, interação de VNFs com funções de rede implementadas em dispositivos proprietários e cenários envolvendo a interoperabilidade de múltiplos *datacenters*. Além de garantir o desempenho de VNFs em infraestruturas compartilhadas, também existe a necessidade de prover os serviços de rede de forma automatizada visando alcançar maior agilidade na entrega destes serviços e consequentemente reduzir custos (Chiosi et al., 2012; Shen et al., 2015; Han et al., 2015; Mijumbi et al., 2016; de Sousa et al., 2019).

Visando aumentar a disseminação e a interoperabilidade das funções virtualizadas de rede, a ETSI (*European Telecommunications Standards Institute*) criou o grupo NFV ISG (NFV *Industry Specification Group*) para propor a padronização de uma arquitetura para a NFV (ETSI, 2014a, 2020a). A Figura 2.2 ilustra o *framework* NFV em alto nível proposto pela ETSI composto por três blocos funcionais principais: *VNFs*, *NFVI* e *NFV Management and Orchestration*, descritos a seguir.

O bloco funcional *Virtualized Network Functions (VNFs)* representa as funções de rede que são implementadas totalmente em software e executadas sobre uma ou mais infraestruturas NFV (NFVI). Cada VNF é então implantada utilizando recursos virtualizados. O bloco *NFV Infrastructure (NFVI)* compreende a variedade de recursos de hardware e software que constituem o ambiente no qual as VNFs são instaladas, gerenciadas e executadas. A abstração dos recursos físicos é realizada por meio de uma camada de virtualização (*Virtualization Layer*) que desacopla

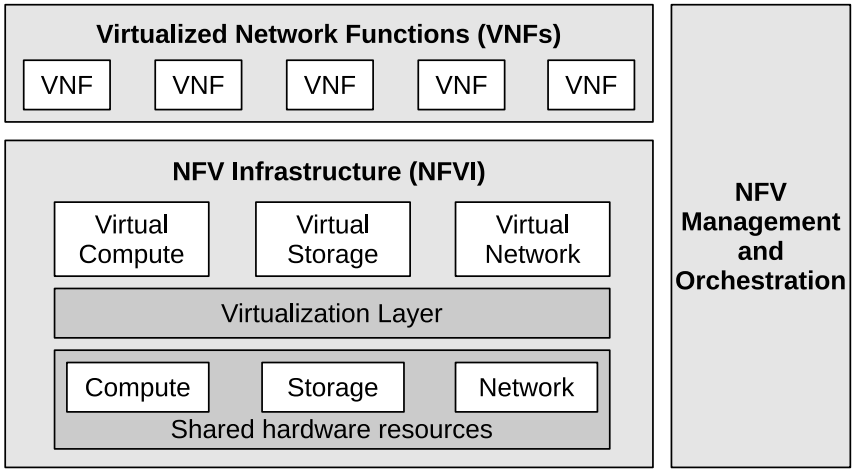


Figura 2.2: *Framework* NFV em alto nível proposto em ETSI (2014a).

os recursos virtuais do hardware subjacente e garante que as VNFs sejam executadas de forma independente do hardware utilizado.

O bloco *NFV Management and Orchestration* da Figura 2.2 é responsável por gerenciar o ciclo de vida das VNFs, como também administrar os recursos de hardware e software que formam a *NFVI*. Diante da complexidade de orquestrar e gerenciar o aprovisionamento de VNFs, a ETSI também propôs a arquitetura de um *framework* para o gerenciamento e orquestração da NFV, denominado NFV-MANO (*NFV - Management and Orchestration*) (Quittek et al., 2014). O *framework* NFV-MANO envolve todos os aspectos do gerenciamento do ciclo de vida das funções virtualizadas de rede que incluem a instanciação de VNFs em máquinas virtuais, abstração dos recursos de hardware, elasticidade, alteração de configurações em tempo de execução, atualização da versão do software das VNFs, bem como encerramento da execução e liberação dos recursos de hardware na NFVI.

A Figura 2.3 detalha o modelo NFV-MANO com seus blocos funcionais e respectivos pontos de referência do ponto de vista da orquestração e gerenciamento de NFV. O bloco *NFV Orchestrator* (NFVO) tem duas responsabilidades principais: o gerenciamento e orquestração dos recursos da *NFV Infrastructure*; e a administração do ciclo de vida dos serviços de rede. O NFVO também utiliza as funções de gerenciamento e orquestração para coordenar a composição de VNFs e formar serviços de rede. A composição permite que um conjunto de VNFs realizem uma tarefa mais complexa, o que implica na instanciação conjunta, a configuração necessária para a conexão entre estas VNFs, bem como alterações dinâmicas de configuração para escalar a capacidade do serviço em tempo de execução caso haja necessidade. O NFVO realiza essas tarefas por meio dos serviços expostos pelas interfaces dos blocos funcionais *VNF Manager* e *Virtualized Infrastructure Manager*.

O bloco funcional *VNF Manager* (VNFM) ilustrado na Figura 2.3 é responsável pelo gerenciamento do ciclo de vida das instâncias de VNFs. A maioria das funções realizadas pelo *VNF Manager* são de finalidade genérica e aplicáveis a qualquer tipo de VNF. Essas funcionalidades são consumidas por elementos do *framework* NFV-MANO e também por entidades externas. Dentre as funções realizadas pelo VNFM pode-se apontar a gerência de VNFs por meio de *templates* de configuração que podem conter informações sobre: endereços IP; atualização da versão do software da VNF; recuperação de falhas; elasticidade; e recursos computacionais. O VNFM também tem acesso a um repositório que armazena as VNFs disponíveis e suas diferentes versões associadas respectivamente aos seus descritores.

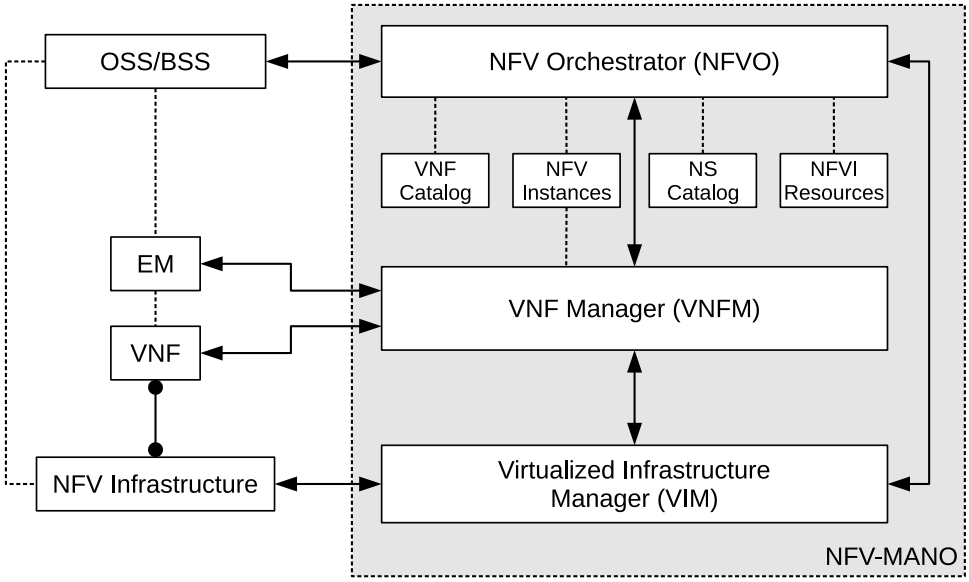


Figura 2.3: O modelo de referência NFV-MANO proposto pela ETSI (Quittek et al., 2014).

O *Virtualized Infrastructure Manager* (VIM) tem a responsabilidade de realizar o controle e gerenciamento dos recursos de hardware e software pertencentes à infraestrutura das operadoras de rede, incluindo recursos para processamento, armazenamento de dados e de rede. Um VIM pode ser genérico e capaz de gerenciar os vários tipos de recursos, ou específico, gerenciar apenas ou recursos de computação, ou armazenamento, ou de rede. O VIM ainda disponibiliza interfaces que possibilitam o gerenciamento dos recursos virtualizados da NFVI e realiza a interação com diferentes hipervisores e controladores de rede.

O *framework* NFV-MANO dispõe de uma base de dados para realizar a tarefa de gerenciamento e orquestração da NFV. Esta base de dados é utilizada pelo NFVO e possui os seguintes elementos. *NS Catalog*: representa o repositório de todos os serviços de rede disponíveis e seus respectivos *templates* de implantação; *NFV Instances*: mantém todas as informações relativas às instâncias de VNFs e serviços de rede, cada VNF e serviço de rede instanciado possui o seu respectivo registro armazenado neste repositório; *VNF Catalog*: representa o repositório com todos os *VNF Packages* disponíveis, cada *VNF Package* contém informações como, imagens de software, descritores das VNFs e arquivos de manifesto; e, *NFVI Resources*: contempla as informações de disponibilidade, reserva e alocação dos recursos da NFVI que são abstraídos pelo VIM. Este repositório tem um papel importante para o NFVO, pois mantém informações dos recursos da NFVI associados aos serviços de rede e VNFs em execução, os quais são utilizados na tomada de decisões durante o processo de gerenciamento e orquestração.

Os demais elementos da Figura 2.3 não são considerados parte do *framework* NFV-MANO porém trocam informações com os blocos funcionais no NFV-MANO. As linhas sólidas com setas representam os principais pontos de referência em andamento no escopo do NFV-MANO. As linhas tracejadas indicam que existem implementações atuais, porém necessitam de extensões para interoperar com a NFV. A associação entre os elementos *VNF* e *NFV Infrastructure* representa um ponto de referência em que há a execução da NFV. (ETSI, 2014a). O *Element Management* (EM) é responsável pelas funcionalidades FCAPS (*Fault, Configuration, Accounting, Performance and Security*) de gerenciamento da VNF correspondente. O elemento *OSS/BSS (Operations Support System / Business Support System)* representa sistemas utilizados por operadores de rede para gerenciar tarefas administrativas tais como, controle de clientes, tarifação, controle de inventário e projeto de redes de comunicação (Quittek et al., 2014).

2.3 *SERVICE FUNCTION CHAINING*

As redes de computadores atuais geralmente empregam múltiplas funções de rede para fornecer serviços aos seus usuários. Nesse contexto, a implantação fim-a-fim de serviços de rede complexos depende da composição de várias funções de rede. Os modelos de implantação de serviços de rede frequentemente são acoplados à topologia da rede e aos recursos físicos de hardware, reduzindo ou eliminando consideravelmente a capacidade de um operador de rede criar novos serviços sob demanda ou modificar serviços e funções de rede existentes.

Esses modelos de implantação de serviços geralmente se apresentam estáticos e pouco flexíveis. Em consequência disso, a modificação de uma ou mais funções de rede em um serviço pode afetar o funcionamento de outras funções daquele mesmo serviço de rede (Halpern e Pignataro, 2015; Yi et al., 2018). O uso da tecnologia NFV permite que as desvantagens associadas ao acoplamento dos recursos físicos e da topologia da rede sejam suprimidas, uma vez que a NFV permite justamente separar as instâncias de software do hardware que as executa.

O *NFV Orchestrator* proposto no modelo NFV-MANO da ETSI pode utilizar funções de gerenciamento e orquestração da infraestrutura NFV disponibilizado pelo VIM para coordenar a composição de VNFs e formar serviços de rede (Quittek et al., 2014). No contexto da NFV, uma função de rede é responsável pelo processamento específico de determinado tráfego da rede e pode operar em diferentes camadas da pilha de protocolos. A instanciação de um conjunto destas funções de rede em uma topologia virtual e o encaminhamento do tráfego em uma determinada sequência através das mesmas forma uma SFC (*Service Function Chaining*). Uma SFC permite construir um serviço de rede composto por determinadas funções de rede e a ordem na qual estas funções são encadeadas (Halpern e Pignataro, 2015; Garcia et al., 2020). A expressão *service chain* também é utilizada para referir-se a uma SFC, ambas são utilizadas como sinônimos no texto.

Diversos esforços vêm sendo realizados para padronizar a composição de serviços de rede (Chiosi et al., 2013; Quittek et al., 2014; IETF, 2020). A IETF (*Internet Engineering Task Force*) criou o IETF SFC WG (*IETF Service Function Chaining Working Group*) (IETF, 2020) para discutir a arquitetura e a padronização de protocolos e extensões que permitam realizar a composição de *service chains* e a sua interação com os elementos do *framework* NFV-MANO. Essencialmente, o modelo definido na arquitetura para a SFC da IETF (Halpern e Pignataro, 2015) faz o encaminhamento do tráfego para próxima função de rede através de um identificador de fluxo em vez de utilizar o roteamento convencional baseado no endereço IP de destino.

Vários componentes lógicos foram projetados para a arquitetura da SFC da IETF. Dentre esses componentes destacam-se *Classifiers*, SFFs (*Service Function Forwarders*), *SFC Proxies* e as próprias funções de rede ilustradas na Figura 2.4. Alguns componentes são responsáveis por restringir o escopo das *service chains* operando sobre o tráfego de entrada e/ou saída das SFCs. Este é o caso dos componentes *Classifier*, SFF e SFC *Proxy* que permitem que o fluxo da *service chain* seja processado pela sequência correta de VNFs. Uma rede ou região de uma rede que possui componentes que implementam funcionalidades no contexto de *service chains* é conhecida como *SFC-enabled Domain* (Halpern e Pignataro, 2015). Cada *SFC-enabled Domain* possui a sua própria rede representada na Figura 2.4 pelo elemento *Network Overlay*. Essa rede é formada por enlaces virtuais e utiliza encapsulamento de tráfego para encaminhar o fluxo da SFC para as respectivas VNFs (Quinn e Nadeau, 2015).

O componente *Classifier* opera na borda de entrada de um *SFC-enabled Domain* e aplica as políticas e restrições definidas pelos operadores da rede. Um *Classifier* intercepta e analisa todo o tráfego baseado nas regras definidas pelo operador de rede para cada *service chain* instanciada. Como resultado da classificação, o tráfego é encapsulado em cabeçalhos de rede

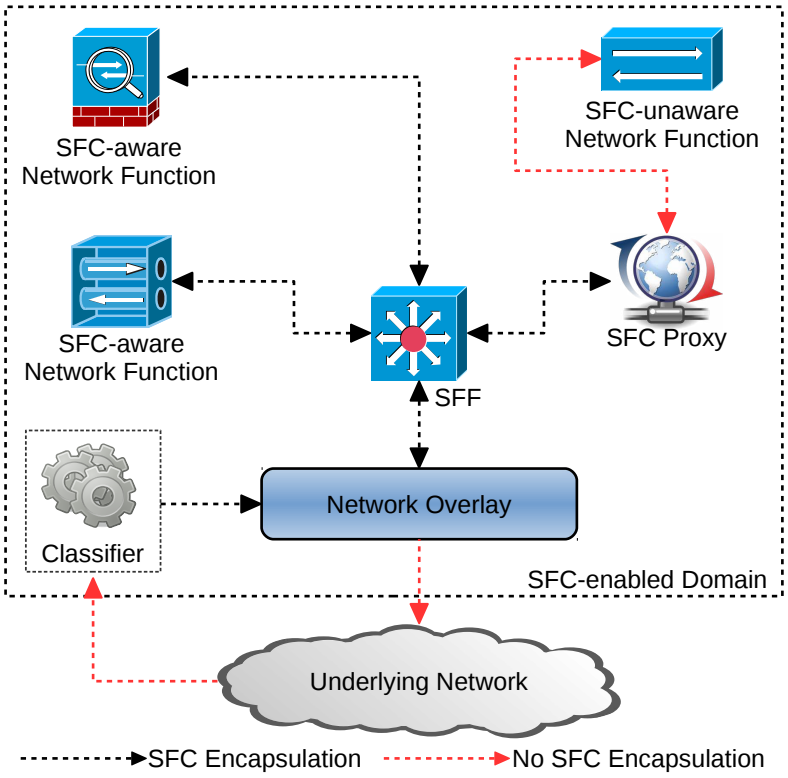


Figura 2.4: Componentes lógicos na arquitetura da SFC da IETF.

que permitem direcionar o fluxo da rede para o SFP (*Service Function Path*) apropriado. Um SFP representa o caminho (*i.e.*, sequência de funções de rede) no qual o tráfego da rede deve atravessar após a classificação, formando um grafo do serviço (Halpern e Pignataro, 2015).

A Figura 2.5 ilustra a associação entre o componente *Classifier* e o SFP por meio de encapsulamento. O encapsulamento permite o direcionamento do conteúdo classificado através de identificadores de fluxo para o *Service Function Path*, que contém neste exemplo três funções de rede. Protocolos usados nesse contexto incluem MPLS (*Multiprotocol Label Switching*), GRE (*Generic Routing Encapsulation*) e VXLAN (*Virtual eXtensible Local Area Network*) (Quinn e Nadeau, 2015).

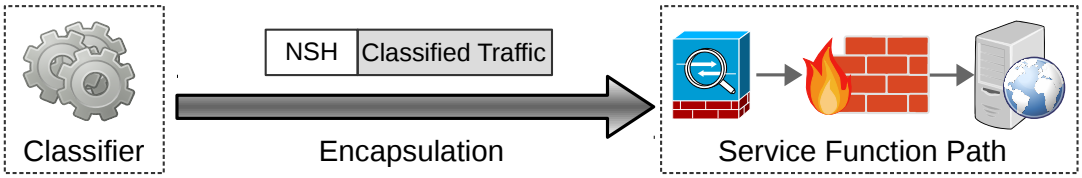


Figura 2.5: Interconexão entre o componente *Classifier* e um SFP.

É importante destacar que o IETF SFC WG vem realizando esforços para definir protocolos para o encapsulamento do tráfego e a troca de informações entre os participantes de um SFP. Um dos protocolos já definido é o NSH (*Network Service Header*) (Quinn et al., 2018). O NSH especifica metadados que indicam a qual SFC um determinado fluxo da rede pertence, bem como identifica as funções de rede que devem processar os dados contidos no campo de carga útil do mesmo. Vale ressaltar que o encapsulamento realizado pelo cabeçalho do NSH não é utilizado para o encaminhamento físico dos fluxos na rede, uma vez que o seu objetivo é carregar informações de contexto da SFC. Sendo assim, o encaminhamento físico dos fluxos da rede continua sob responsabilidade dos protocolos subjacentes da rede.

Um *SFC-enabled domain* requer que os componentes da rede sejam capazes de manipular as informações contidas no cabeçalho da SFC. Contudo, componentes da rede originalmente incapazes de manipular o cabeçalho da SFC podem ser conectados a um SFC *Proxy* para manipulação externa do cabeçalho, conforme ilustrado na Figura 2.4. O componente SFC *Proxy* possibilita que funções de rede tradicionais e sem o conhecimento do cabeçalho da SFC sejam adicionadas à *service chain*.

O SFF da Figura 2.4 é responsável pelo encaminhamento do fluxo recebido da rede através do componente *Classifier* para uma ou mais funções de rede – virtuais ou físicas – de acordo com as informações transmitidas no cabeçalho da SFC. Após a função de rede processar o tráfego, este retorna novamente ao SFF que então realiza o encaminhamento para a próxima função de rede do SFP. Uma SFC é executada completamente a partir do momento em que o tráfego tenha sido processado por todas as funções de rede do SFP. O SFF também é responsável por remover o encapsulamento da SFC e introduzir o tráfego novamente na rede subjacente depois que a última VNF do SFP processou o tráfego da rede.

A reclassificação do tráfego também pode ocorrer enquanto o fluxo atravessa uma SFC indicando que a sequência de VNFs originalmente definida foi modificada. O elemento da rede onde ocorre a reclassificação é chamado de *Branching Node*. Isso possibilita que mais de um caminho (*i.e.*, SFP) possa ser utilizado a partir de uma determinada função de rede (Halpern e Pignataro, 2015). Esse é um caso comum quando o fluxo de rede atravessa sistemas de segurança, como por exemplo, um IDS (*Intrusion Detection System*) que pode redirecionar o fluxo da rede dependendo do resultado de sua análise ou conduzir ao serviço de destino (Alharbi et al., 2017).

As SFCs também podem apresentar fluxos com características simétricas ou assimétricas. SFCs simétricas indicam que o processamento do fluxo de retorno (*i.e.*, sequência de VNFs pelas quais o fluxo passa) deve ser exatamente o inverso do fluxo de envio. Por outro lado, SFCs assimétricas não definem qualquer relação entre os fluxos de envio e retorno (Quinn e Nadeau, 2015). Também existem SFCs híbridas, nas quais o tráfego reverso é necessário apenas em um conjunto de funções de rede (Halpern e Pignataro, 2015).

2.4 CONCLUSÃO

Este capítulo apresentou o paradigma NFV de virtualização de funções de rede. As funções virtualizadas de rede são uma alternativa aos *middleboxes* tradicionais, permitindo que funcionalidades diversas sejam implementadas em software e virtualizadas em hardware de prateleira. Serviços complexos de rede podem ser criados por uma composição de funções virtualizadas de rede formando SFCs que trazem flexibilidade e agilidade na entrega de novos serviços de rede. O capítulo seguinte apresenta a proposta de um *framework* holístico para a composição de SFCs.

3 COMPOSIÇÃO HOLÍSTICA DE SERVIÇOS DE REDE

Este capítulo descreve o *framework Holistic-Composer* para a composição e o gerenciamento do ciclo de vida de serviços de rede. Um serviço de rede é uma *service chain* que pode ser orquestrada por diferentes plataformas NFV. Inicialmente, é apresentada a arquitetura do *framework* proposto. Em seguida, são abordados os facilitadores NFV utilizados no desenvolvimento do *framework*, os detalhes de implementação e a avaliação da abordagem proposta. Por fim, é realizada uma comparação com outras abordagens.

3.1 O *FRAMEWORK HOLISTIC-COMPOSER*

O *Holistic-Composer* permite a definição e o gerenciamento do ciclo de vida de uma *service chain* que pode ser orquestrada por diferentes plataformas NFV. Um dos objetivos principais do *Holistic-Composer* é reduzir a complexidade de diferentes plataformas NFV na composição e gerenciamento *service chains*, de forma que os operadores de rede não necessitem de conhecimentos específicos de todas as tecnologias e particularidades das respectivas plataformas NFV. Um *framework* é proposto para o *Holistic-Composer* como uma solução genérica e emprega os padrões da especificação NFV-MANO da ETSI, permitindo o seu uso nas diferentes plataformas NFV compatíveis com o NFV-MANO e em múltiplas aplicações cliente. O objetivo final do *framework* proposto é simplificar a composição de *service chains* de modo que o operador de rede compõe a SFC apenas informando a sequência de VNFs e, quando necessário, as conexões entre as mesmas.

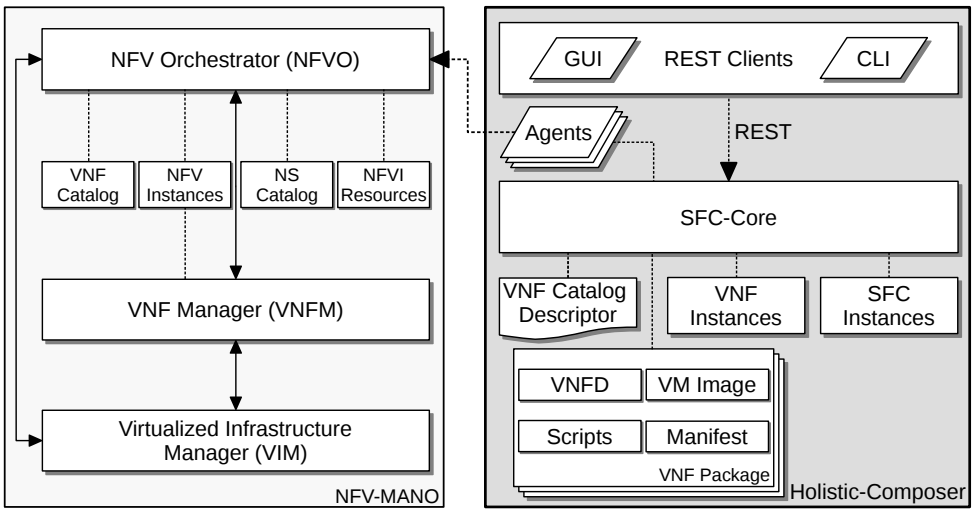


Figura 3.1: Arquitetura proposta para o *framework Holistic-Composer*.

A Figura 3.1 ilustra a arquitetura do *framework* proposto. O *framework* é dito holístico no sentido que abstrai as especificidades de diferentes plataformas NFV através de uma API genérica e única. O *framework* NFV-MANO da ETSI também é ilustrado na Figura 3.1, visto que a arquitetura proposta para o *Holistic-Composer* complementa o NFV-MANO sem a exigência de modificações e possibilita que as duas arquiteturas possam evoluir em paralelo. É possível observar que existe um ponto de comunicação entre as duas arquiteturas: *Agents* → *NFVO*. Os agentes de comunicação (*Agents*) foram utilizados para abstrair as especificidades de interação

com as diferentes plataformas NFV. Para cada orquestrador NFV existe um agente de comunicação responsável por implementar as respectivas interações e particularidades.

O *SFC-Core* é o principal componente do *framework* proposto e foi desenvolvido com o propósito de conduzir e validar as operações requisitadas pelas aplicações cliente. Este componente possibilita abstrair a complexidade na definição e gerenciamento do ciclo de vida das *service chains*. O *SFC-Core* disponibiliza uma interface de comunicação padronizada que utiliza o modelo REST (*REpresentational State Transfer*) e permite que os mais variados tipos de aplicações cliente possam interagir com o *framework* proposto. Essas aplicações são ilustradas na Figura 3.1 como *REST Clients* e incluem aplicações que usam GUI (*Graphical User Interface*) e CLI (*Command Line Interface*).

O *SFC-Core* também gerencia o repositório local de VNFs representado pelo elemento *VNF Package* na Figura 3.1. O formato do *VNF Package* segue o modelo da ETSI (Kojukhov et al., 2017) permitindo a interoperabilidade do *framework* proposto com o maior número de plataformas NFV compatíveis com o NFV-MANO. Cada *VNF Package* possui um descritor da VNF (*VNFD*) e os respectivos *scripts* que devem ser executados ao instanciar a VNF na infraestrutura NFV. O *VNF Package* também pode conter a imagem do software que irá executar a VNF (*i.e.*, *VM Image*), ou, a mesma pode ser utilizada a partir do repositório do NFVO. O elemento *Manifest* representa o descritor do *VNF Package* que contempla informações como o nome da VNF, o desenvolvedor, a versão e a data de liberação.

O elemento *VNF Catalog Descriptor* da Figura 3.1 é utilizado pelo *SFC-Core* para gerenciar informações de metadados dos *VNF Packages* armazenados no repositório local. Os metadados incluem um identificador único, nome, localização, descrição, tipo da função e categoria da VNF. O tipo da função representa a plataforma que executa a VNF (*e.g.*, Click-on-OSv ou Linux). O Click-on-OSv, por exemplo, necessita de um *Element Management* específico do *framework* NFV-MANO e requer etapas extras para a configuração da VNF. O tipo da função também possibilita que VNFs programadas em diferentes plataformas de execução e com particularidades diversas sejam contempladas pelo *framework* proposto.

A definição da categoria da VNF no *VNF Catalog Descriptor* permite diferenciar a classe de serviço que aquele *VNF Package* oferece, como por exemplo, *firewall*, balanceador de carga, *proxy* reverso, tunelamento, dentre outros. As classes de serviços oferecidas permitem que o componente *SFC-Core* seja integrado com outras soluções capazes de sugerir possíveis VNFs para a composição, como também pode viabilizar que serviços de rede sejam compostos de forma automatizada. Também é proposto um catálogo (*VNF Instances*) com informações das VNFs gerenciadas pelo *Holistic-Composer* para permitir a interoperabilidade com diferentes plataformas NFV e demais soluções para SFC. Este catálogo permite diferenciar as VNFs instanciadas a partir de ferramentas de terceiros daquelas gerenciadas pelo *framework Holistic-Composer*.

O catálogo *SFC Instances* da Figura 3.1 mapeia as instâncias de *service chains* que estão executando no orquestrador NFV. As informações das VNFs instanciadas e que fazem parte da composição das SFCs também são mantidas por este catálogo. A utilização dos catálogos de *VNF Packages*, *VNF Instances*, *SFC Instances* em conjunto com o mapeamento das instâncias de VNFs e SFCs possibilita a interoperabilidade com diferentes plataformas NFV e dispensa a modificação do *framework* NFV-MANO e suas implementações.

3.2 IMPLEMENTAÇÃO DA ARQUITETURA PROPOSTA

A implantação de serviços de rede em infraestruturas existentes pode ser facilitada através da utilização de ferramentas de virtualização disponíveis atualmente. Para a ETSI, ferramentas que contribuem no desenvolvimento e implantação de NFV são chamadas de

facilitadores de NFV (*NFV Enablers*) (Chiosi et al., 2012). Embora nem todos os facilitadores NFV tenham sido desenvolvidos especificamente para a NFV (*e.g.*, OpenStack), esses podem ser estendidos e aplicados na NFV para atender a diferentes requisitos que podem compreender desde a virtualização dos recursos físicos até o gerenciamento do ciclo de vida das funções virtualizadas de rede. A implementação do *framework* proposto utiliza-se de facilitadores NFV como OpenStack, Click-on-OSv e Tacker nas tarefas de composição e gerenciamento do ciclo de vida das *service chains*.

O OpenStack (OpenStack, 2020a) é uma plataforma para a implementação de infraestruturas para computação em nuvem. Sua arquitetura é baseada em componentes diversos que podem ser implantados em diferentes servidores físicos e cada um responsável por uma função específica (*e.g.*, nós de computação, rede, armazenamento). Estes componentes integrados disponibilizam uma infraestrutura de computação em nuvem completa e modular. O OpenStack pode ser utilizado no contexto da NFV para a virtualizar os recursos físicos utilizados pelas VNFs e atuar como um VIM no ambiente NFV. A decisão de utilizar o OpenStack como a implementação do VIM se justifica por ser uma solução estável e com suporte a uma grande quantidade de hardware, além de ser uma plataforma modular e bastante difundida na comunidade NFV. Vale lembrar que o *framework* proposto foi idealizado para executar com diferentes implementações do VIM embora apenas o OpenStack seja apresentado neste momento.

O Click-on-OSv (da Cruz Marcuzzo et al., 2017) é uma plataforma capaz de executar funções de rede baseadas no *Click Modular Router* (Kohler et al., 2000) sobre o *unikernel* OSv (OSv, 2020), permitindo a criação de VNFs minimalistas de alto desempenho. O Click-on-OSv também possui suporte a múltiplos hipervisores (*e.g.*, Xen, KVM, VMWare e VirtualBox) e possibilita o gerenciamento das funcionalidades FCAPS das VNFs através de uma interface Web ou API REST. O DPDK (*Data Plane Development Kit*) (Linux Foundation, 2020a) também é suportado pelo Click-on-OSv. O DPDK é composto por um conjunto de bibliotecas e *drivers* que permitem obter alto desempenho no processamento do tráfego da rede. A capacidade do Click-on-OSv executar em múltiplos hipervisores também possibilita que a implementação da arquitetura proposta para o *Holistic-Composer* seja integrada e aproveitada por uma quantidade maior de plataformas NFV.

O Tacker (Tacker, 2020) é um projeto oficial da plataforma OpenStack e tem como objetivo desenvolver funcionalidades de *VNF Manager* e *NFV Orchestrator* genéricos para gerenciar e operar serviços de rede e VNFs em uma infraestrutura NFV. O Tacker é baseado na especificação NFV-MANO da ETSI e fornece funcionalidades para orquestrar serviços de rede usando VNFs de ponta a ponta. As funcionalidades incluem desde a administração básica de VNFs até o gerenciamento do ciclo de vida dos serviços de rede e o monitoramento das instâncias de VNFs.

A implementação do Tacker utiliza a linguagem descritiva TOSCA (*Topology and Orchestration Specification for Cloud Applications*) (Li e Crandall, 2017) para descrever os serviços de rede e as instâncias das VNFs. Os descritores de VNFs (VNFDs) são armazenados em um catálogo e utilizados pelo mecanismo de orquestração OpenStack Heat (Heat, 2020) que é responsável por instanciar e destruir as VNFs e serviços de rede. A virtualização da rede é gerenciada pelo Tacker através do OpenStack e utiliza o Open vSwitch (Linux Foundation, 2020b) que implementa o protocolo OpenFlow (ONF, 2020). O Open vSwitch executa as funcionalidades dos componentes SFF e SFC *Proxy* descritos na Seção 2.3. O Open vSwitch encaminha o fluxo da rede entre as VNFs da SFC utilizando os protocolos MPLS e NSH.

Um operador de rede necessita de conhecimentos específicos de plataforma para compor e gerenciar as *service chains*, tanto para o uso do Tacker como de outros orquestradores NFV descritos na Seção 3.5. Além do conhecimento operacional, a compreensão das linguagens

de descrição (*e.g.*, TOSCA/YAML ou NETCONF/YANG) utilizadas pelas plataformas NFV se torna indispensável. O operador de rede ainda necessita conhecer as particularidades e os conceitos de SFC descritos na Seção 2.3. O descritor utilizado pelo OpenStack Tacker para formar serviços de rede através da composição de VNFs é o VNFFGD (VNF *Forwarding Graph Descriptor*). Um VNFFGD define os diferentes SFPs (*Service Function Paths*), VNFs, protocolo de encapsulamento e as políticas de classificação de tráfego da SFC.

A implementação de todos os componentes para o protótipo da arquitetura do *framework Holistic-Composer* foi realizada através da linguagem Python. O componente *SFC-Core* ilustrado na Figura 3.1 fornece uma interface de comunicação através do modelo REST que foi implementada com o uso da biblioteca *Flask* da linguagem Python. Clientes REST dos mais variados tipos podem ser implementados para interagir com o *SFC-Core* através dessa API REST. A API é genérica e possibilita compor e gerenciar o ciclo de vida de *service chains* em diferentes plataformas NFV. Todas as informações que os clientes do *framework* proposto necessitam permanecem armazenadas em uma base de dados local que é acessada e gerenciada pelo componente *SFC-Core*. Nesta base de dados são armazenadas informações sobre os *VNF Packages*, *VNF Catalog Descriptor*, *VNF Instances* e *SFC Instances*. Os *VNF Packages* e seus recursos são armazenados em diretórios com identificadores únicos gerados pelo *SFC-Core*.

A implementação do componente *SFC-Core* também viabiliza que os descritores de VNFs contidos nos *VNF Packages* possam utilizar os formatos JSON (*JavaScript Object Notation*) ou TOSCA/YAML. Isso se deve ao fato de que orquestradores NFV como o Tacker e o Open Baton descrito na Seção 3.5, requerem o envio dos descritores em formato JSON. Além disso, cada registro do elemento *VNF Instances* armazena o identificador único do *VNF Package* utilizado e os identificadores do VNFD e da instância da VNF gerados pelo orquestrador NFV. As informações armazenadas são utilizadas para destruir as VNFs e seus descritores no momento em que a *service chain* é removida do orquestrador NFV.

O elemento *SFC Instances* armazena informações sobre as instâncias de *service chains* que estão ativas no orquestrador NFV e contempla informações como o identificador local e único da SFC, a lista das instâncias de VNFs envolvidas na composição, bem como o descritor e o identificador da instância do VNFFG (VNF *Forwarding Graph*) que está em execução no OpenStack Tacker. Também foi implementado um agente de comunicação para a troca de informações entre o *framework* proposto e a API REST disponibilizada pelo Tacker. Este agente de comunicação foi implementado com o auxílio da biblioteca *Requests* da linguagem Python. Cada agente de comunicação é responsável por traduzir as abstrações do *SFC-Core* para a interface de comunicação específica do respectivo orquestrador NFV.

Ainda foi codificada uma aplicação cliente na linguagem Python para avaliar a implementação do *framework* proposto. Esta aplicação executa através de CLI (*Command Line Interface*) e permite compor e gerenciar *service chains* de maneira simples e objetiva. Também foi implementado o componente *Element Management* da especificação do NFV-MANO para interagir com o Click-on-OSv, uma vez que o *VNF Manager* implementado pelo Tacker é genérico e pode gerenciar apenas o contexto do *unikernel* OSv.

3.3 COMPOSIÇÃO DE *SERVICE CHAINS*

Tendo em vista a complexidade para abstrair a composição de VNFs para formar SFCs, a Figura 3.2 demonstra em alto nível a sequência de troca de mensagens requerida entre os componentes implementados no protótipo e o orquestrador NFV OpenStack Tacker durante o processo de composição das SFCs. As mensagens estão indexadas em ordem numérica para facilitar o entendimento da lógica implementada.

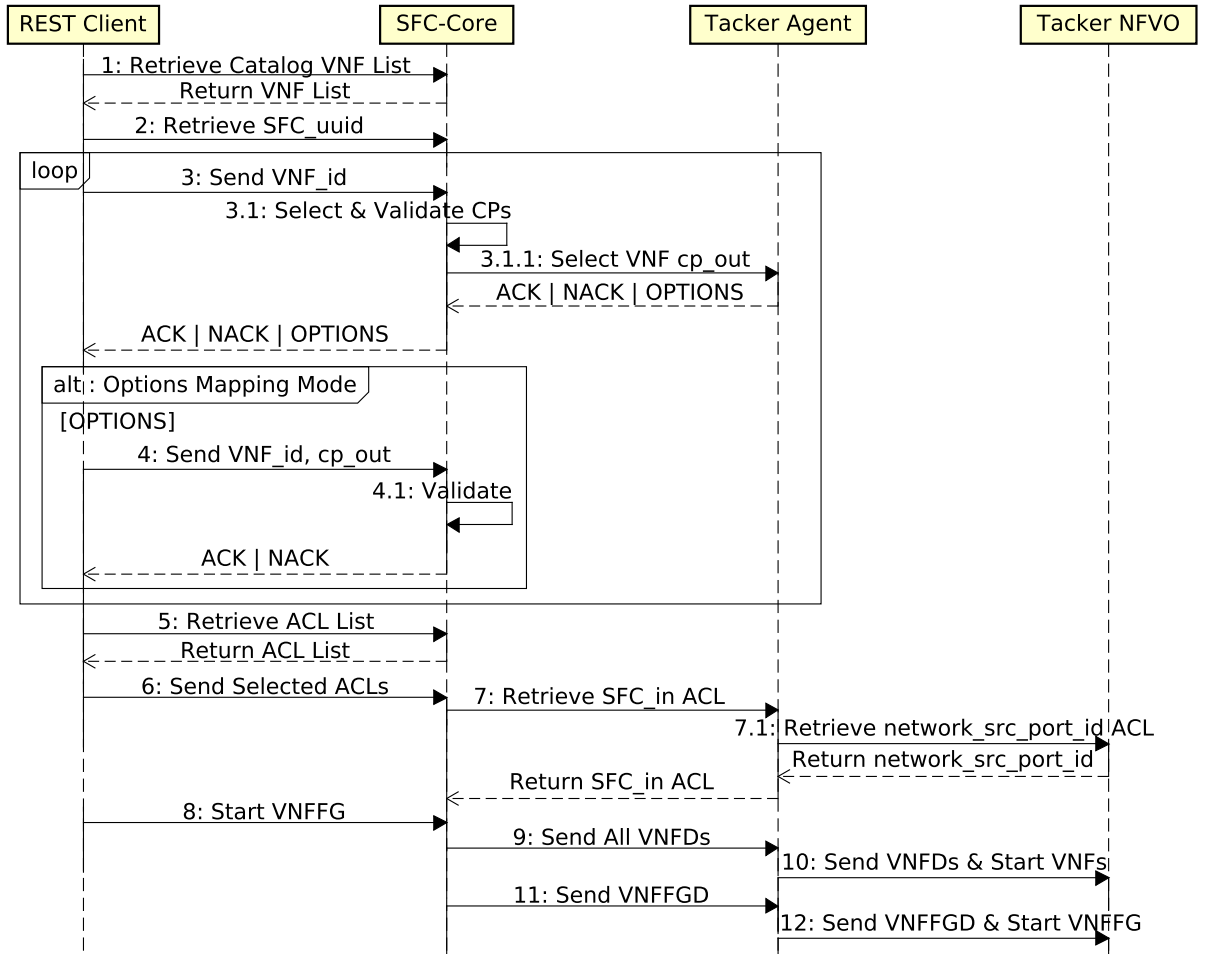


Figura 3.2: Troca de mensagens entre componentes implementados no protótipo e o NFVO Tacker durante a composição de uma SFC.

Na Figura 3.2 são apresentados os quatro componentes principais envolvidos na composição de *service chains*. O componente *REST Client* representa a aplicação cliente que interage com a API REST disponibilizada pelo componente *SFC-Core*. O *SFC-Core* implementa a lógica para compor as SFCs abstraindo as especificidades dos diferentes orquestradores NFV e valida as entradas de dados informadas pelo usuário. O componente responsável por tratar as particularidades do orquestrador NFV é representado pelo elemento *Tacker Agent* e o componente *Tacker NFVO* corresponde à plataforma NFV OpenStack Tacker. O protótipo implementado também permite compor SFCs utilizando VNFs implementadas no modelo tradicional (*e.g.*, *Proxy* Reverso em Linux) em conjunto com VNFs que executam sobre o Click-on-OSv.

A mensagem "*1: Retrive Catalog VNF List*" enviada pelo componente *REST Client* ao *SFC-Core* faz uma requisição solicitando a lista de VNFs disponíveis para a composição. O *SFC-Core* executa uma consulta ao catálogo de *VNF Packages* e responde a requisição informando a lista completa de VNFs existentes neste catálogo. Em seguida, o componente *REST Client* envia a mensagem "*2: Retrieve SFC_uuid*" ao elemento *SFC-Core* para obter um identificador universal único usado para identificar a composição da *service chain* que está em andamento em todas as trocas de mensagens seguintes. Isto permite que múltiplas composições de *service chains* sejam executadas de forma concorrente.

O operador de rede seleciona uma das VNFs disponíveis no catálogo que é então submetida pelo *REST Client* ao *SFC-Core* através da mensagem "*3: Send VNF_id*". Esta mensagem carrega o identificador único que representa a VNF selecionada pelo operador de

rede. O *SFC-Core* então recupera as informações do respectivo descritor da VNF do repositório local e em seguida seleciona e valida os *Connection Points* (CP) da VNF automaticamente. A especificação do NFV-MANO (Quittek et al., 2014) define que um *Connection Point* pode representar uma interface física ou virtual e fornece a conectividade de rede necessária para a troca de mensagens entre funções e serviços de rede.

A operação de seleção e validação dos *Connections Points* é desencadeada no *SFC-Core* através da mensagem "*3.1: Select & Validate CPs*". Os passos descritos a seguir reproduzem a seleção e validação dos *Connection Points* apropriados. Inicialmente o componente *SFC-Core* busca por um *Connection Point* que está associado ao mesmo *Virtual Link* (*i.e.*, nome da sub-rede) do *Connection Point* de saída da VNF anterior naquela SFC. Caso um *Connection Point* com tais características for encontrado, o mesmo é selecionado para receber o tráfego de entrada da VNF que está sendo incluída na SFC. Caso não for encontrada nenhuma relação de *Virtual Links* entre os *Connection Points* das duas VNFs, uma mensagem com o estado *NACK* é retornada ao *REST Client* indicando que aquela VNF não pode ser incluída na SFC.

Caso a VNF que está sendo incluída seja a primeira ou até mesmo a única da composição da SFC, então o primeiro *Connection Point* válido especificado no descritor daquela VNF é selecionado. Como os orquestradores NFV possuem redes virtualmente isoladas para o processamento do tráfego e para o gerenciamento das funcionalidades FCAPS das VNFs, nem todos os *Connection Points* são válidos para compor uma SFC. *Connection Points* associados às redes de gerenciamento das VNFs são considerados inválidos e por isso não são usados na composição.

As regras para a seleção e validação de *Connection Points* de saída das funções de rede dependem diretamente do orquestrador NFV que está sendo usado para compor a SFC. Na situação em que a VNF possui apenas um *Connection Point* e as regras para a seleção do *CP* de entrada são satisfeitas, então este mesmo *Connection Point* é selecionado para tráfego de saída da função de rede e uma mensagem de confirmação (*i.e.*, *ACK*) é enviada ao *REST Client*. Contudo, outros *Connection Points* podem existir no descritor de uma VNF. Neste caso, o *SFC-Core* não pode inferir o *Connection Point* de saída da VNF, pois as especificidades do orquestrador NFV estão implementadas no agente de comunicação. A mensagem "*3.1.1: Select VNF cp_out*" é enviada para o agente de comunicação *Tacker Agent* que então seleciona o *Connection Point* com base nas particularidades da plataforma Tacker.

Diferentes regras e particularidades podem ser aplicadas por outros agentes de comunicação que implementam a interação do *SFC-Core* com os orquestradores NFV. Outros orquestradores poderiam, por exemplo, possibilitar a composição de uma SFC em diferentes sub-redes, ou até mesmo permitir a composição de SFCs utilizando VNFs com diferentes *Connection Points* associados ao mesmo *Virtual Link*. O agente de comunicação aplica uma das seguintes regras caso múltiplos *Connection Points* estejam associados ao mesmo *Virtual Link*: **a)** se existir apenas um *Connection Point* válido, o agente de comunicação seleciona este *Connection Point* como interface de saída da VNF e responde com uma mensagem de confirmação (*i.e.*, *ACK*); ou **b)** se existir mais de um *Connection Point* válido que possa ser associado na composição da SFC, a mensagem *OPTIONS* e a lista de *Connection Points* válidos e seus respectivos *Virtual Links* é submetida do agente de comunicação para o *SFC-Core* que então encaminha a mensagem para o *REST Client*. Esta mensagem indica que o operador da rede deve selecionar o *Connection Point* desejado manualmente e o bloco alternativo "*alt: Options Mapping Mode*", também ilustrado na Figura 3.2, é executado.

Durante a execução do bloco alternativo, o componente *REST Client* envia uma mensagem contendo o *Connection Point* selecionado pelo operador da rede ao componente *SFC-Core*. O *SFC-Core* então valida o *Connection Point* de saída da VNF e submete uma

mensagem de resposta indicando se foi possível incluir a VNF na SFC (*i.e.*, *ACK*), ou, se o *Connection Point* enviado na mensagem está incorreto (*i.e.*, *NACK*). O padrão de troca de mensagens, representado pelo bloco "*loop*" na Figura 3.2, é executado repetidamente até que todas as VNFs foram incluídas na composição da *service chain*.

A seleção do *Connection Point* de saída das VNFs depende diretamente das características particulares de cada orquestrador NFV, uma vez que cada NFVO implementa a sua própria abordagem para formar SFCs. VNFs que operam como *Branching Nodes* nas SFCs também podem compartilhar múltiplos *Connection Points* em um mesmo *Virtual Link*. Outra possibilidade é uma VNF que opera como *Branching Node* utilizar diferentes *Virtual Links* para executar essa tarefa (*i.e.*, um para cada *Connection Point*). Cada orquestrador NFV também apresenta restrições e limitações específicas para compor as SFCs. Por exemplo, a versão *Pike* do OpenStack Tacker – utilizada durante a implementação e avaliação do *framework* proposto – não possibilita a execução de SFCs que atravessam mais do que uma sub-rede (*i.e.*, todos os *Connection Points* devem estar conectados ao mesmo *Virtual Link*). A mesma versão do OpenStack Tacker também não permite criar *Service Function Paths* com VNFs que possuem dois ou mais *Connection Points* compartilhando o mesmo *Virtual Link* na mesma SFC.

Após todas as VNFs serem incluídas corretamente na SFC, a aplicação *REST Client* envia a mensagem "*5: Retrieve ACL List*" ao *SFC-Core* solicitando as restrições de ACL (*Access Control List*) disponíveis. As ACLs definem as políticas e restrições que devem ser aplicadas ao fluxo de rede pelo componente *Classifier* para encapsular e direcionar o tráfego da rede para o *Service Function Path* apropriado. O OpenStack Tacker permite encapsular o tráfego da rede por meio dos protocolos MPLS e NSH implementados no Open vSwitch.

O operador de rede então seleciona as restrições da ACL e informa seus respectivos valores (*e.g.*, *ip_proto: 6*, *destination_port_range: 80-80* e *ip_dst_prefix: 10.10.0.5/32*). A mensagem "*6: Send Selected ACLs*" representa o envio destas restrições ao *SFC-Core* que submete a mensagem "*7: Retrieve SFC_in ACL*" ao *Tacker Agent*. Esta mensagem visa recuperar o identificador da porta do Open vSwitch que será a origem do tráfego da SFC. Este identificador é uma particularidade do Open vSwitch para configurar a ACL do classificador da SFC. O *Tacker Agent* recupera o identificador através da mensagem "*7.1: Retrieve network_src_port_id ACL*" usando a API REST do *Tacker NFVO*. O identificador é adicionado na ACL do classificador assim que o *SFC-Core* recebe a resposta do componente *Tacker Agent*.

Para instanciar a SFC, a aplicação *REST Client* envia a mensagem "*8: Start VNFFG*" para o *SFC-Core*. Na sequência, o *SFC-Core* envia todos os descritores das VNFs selecionadas ao agente de comunicação *Tacker Agent* através da mensagem "*9: Send All VNFDs*". O *Tacker Agent* então submete e instancia todos estes descritores de VNFs ao *Tacker NFVO* através de uma API REST. As operações de envio dos VNFDs e a instanciação das VNFs no OpenStack Tacker são representadas pela mensagem "*10: Send VNFDs & Start VNFs*". O componente *SFC-Core* então envia o descritor da SFC (*i.e.*, *VNFFGD*) ao *Tacker Agent* por meio da mensagem "*11: Send VNFFGD*" após todas as VNFs terem sido instanciadas com sucesso. Por fim, o *Tacker Agent* envia o VNFFGD ao *Tacker NFVO* e solicita a instanciação da SFC através da mensagem "*12: Send VNFFGD & Start VNFFG*". Ainda foram implementadas funcionalidades que permitem cancelar a composição da SFC e a execução das VNFs (*i.e.*, *rollback*) caso ocorram erros durante o processo de instanciação da SFC ou de suas VNFs.

3.4 AVALIAÇÃO DA ARQUITETURA PROPOSTA

Uma avaliação de desempenho foi executada para analisar o tempo que o *Holistic-Composer* necessita para compor SFCs nos diferentes cenários descritos abaixo. As avaliações

foram executadas usando um computador com a seguinte configuração: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz com 4 núcleos; 12 GiB de RAM DDR4 a 2133 MHz; Linux Ubuntu 16.04.3 executando o *kernel* 4.13.0-37-generic x86_64; e, Apache HTTP Server 2.4.18 configurado com o módulo *libapache2-mod-wsgi-py3*. Este módulo permite o encaminhamento das requisições recebidas pelo servidor Apache ao componente *SFC-Core* através de uma interface WSGI (*Web Server Gateway Interface*). A biblioteca *Flask* conecta o *SFC-Core* à interface WSGI.

O *SFC-Core* foi configurado para processar 4 requisições simultâneas (*i.e.*, 4 processos) e com isso paralelizar a composição das SFCs. O sistema de memória *cache* distribuída *memcached* versão 1.4.25 (Memcached, 2020) foi utilizado para realizar a comunicação inter-processos durante a composição das SFCs. O banco de dados MongoDB versão 2.6.10 foi empregado para armazenar os dados manipulados pelo *SFC-Core*.

Também foi implementado um cliente REST para realizar os experimentos que envolvem a composição simultânea das SFCs. A aplicação cliente apenas envia requisições REST ao *SFC-Core* que é responsável pela lógica da composição e gerenciamento do ciclo de vida das SFCs. O tempo gasto em cada composição foi computado no código do cliente REST e as medidas foram realizadas no início da primeira requisição e no recebimento da resposta da última requisição de cada composição. Todos os experimentos foram executados 100 vezes e as médias são apresentadas; o intervalo de confiança é de 95%.

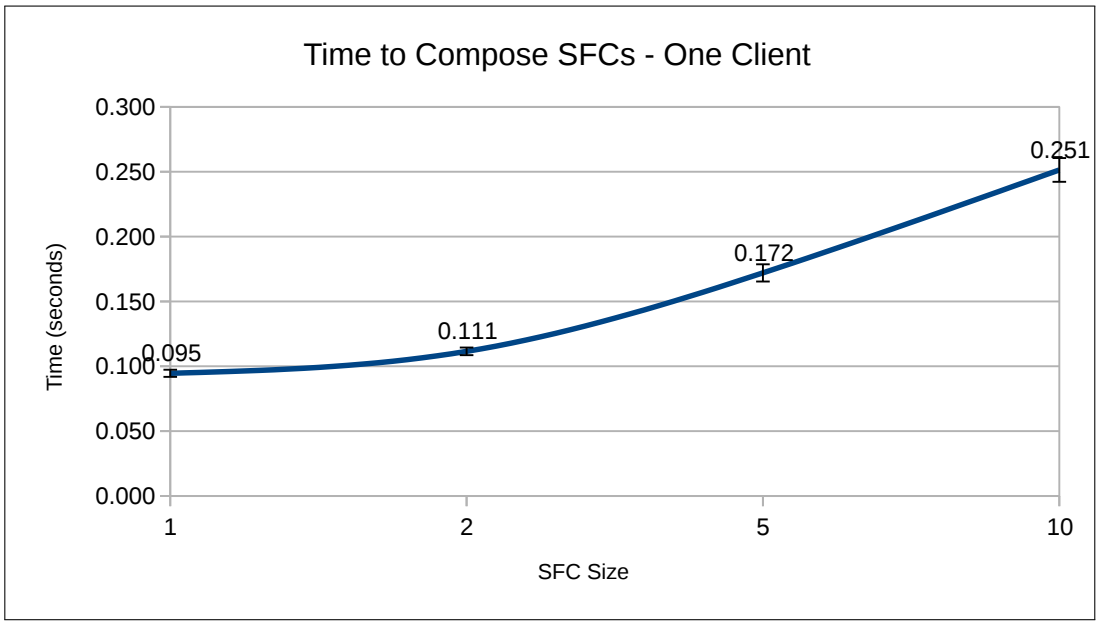


Figura 3.3: Tempo para compor SFCs com diferentes tamanhos.

A Figura 3.3 ilustra o tempo gasto por um único cliente REST para compor SFCs sequencialmente com tamanho de 1, 2, 5 e 10 VNFs. É possível observar que após adicionar a segunda VNF são necessários em média aproximadamente 17,5ms extras para incluir cada uma das VNFs seguintes. Também pode-se verificar que o protótipo implementado necessitou em média 251ms para compor uma SFC com tamanho de 10 VNFs. Considerando o intervalo de confiança de 95%, é possível estimar que a variação presente na amostra para compor SFCs com tamanhos 1, 2, 5 e 10 VNFs foi respectivamente 2,7ms, 2,9ms, 6,6ms e 9,1ms.

A Figura 3.4 mostra o tempo necessário para compor diferentes quantidades de SFCs cada uma com tamanho de 10 VNFs. Foram usadas duas abordagens: *Sequential* e *Holistic*. Na abordagem *Sequential*, o cliente REST compõe o número de SFCs sequencialmente e necessitou

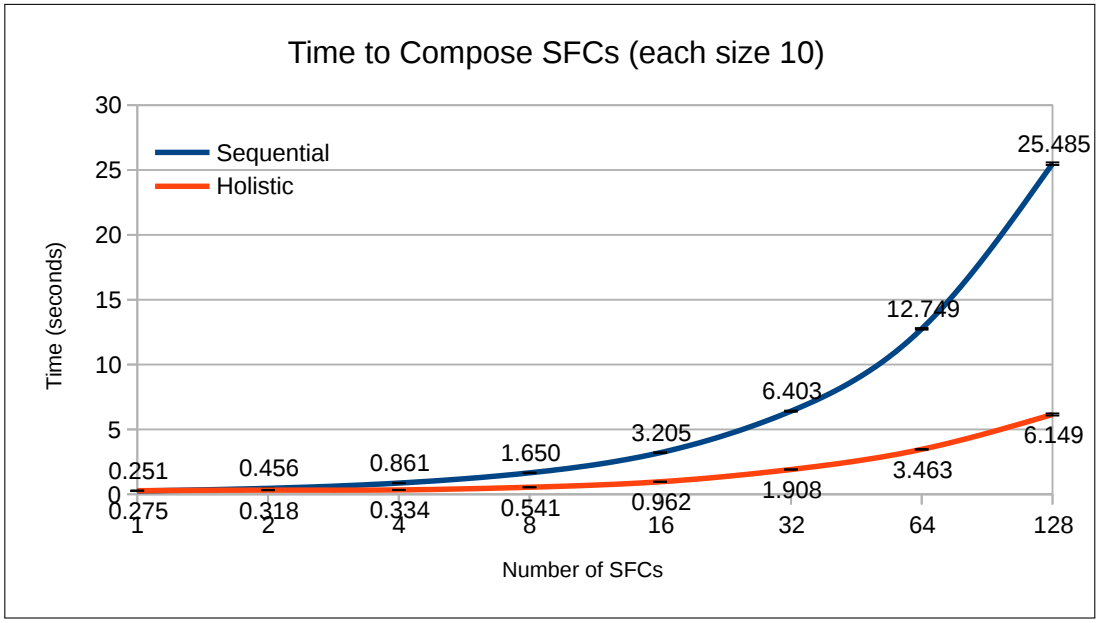


Figura 3.4: Tempos para compor diferentes quantidades de SFCs.

em média 25,485 segundos para compor 128 SFCs neste experimento. A variação do tempo de execução da amostra para compor as 128 SFCs foi de 100ms observando-se o intervalo de confiança de 95%. A abordagem *Holistic* executou a composição das 128 SFCs de forma paralela e necessitou em média 6,149 segundos para realizar a mesma tarefa. Considerando o intervalo de confiança de 95%, a variação da amostra obtida com o uso da abordagem *Holistic* foi de 79ms. Assim, é possível constatar que neste experimento a abordagem *Holistic* foi aproximadamente 4,145 vezes mais eficiente se comparada com o tempo de execução da abordagem *Sequential*.

Também foi realizado um experimento para avaliar o *speedup* do *framework* implementado utilizando o teste de escalabilidade forte que considera o tamanho do problema fixo (*i.e.*, número de SFCs) e aumenta a quantidade de recursos (*i.e.*, número de processos). Neste experimento foi avaliada a composição simultânea de 64 SFCs cada uma com tamanho de 10 VNFs. O número de processos usados para atender as requisições em paralelo foi incrementado de forma exponencial durante a execução dos diferentes cenários do experimento. A Figura 3.5 ilustra o gráfico de *speedup* do *framework* proposto. É possível observar que o *speedup* se mostra satisfatório até o limite de 8 processos executados em uma máquina com um processador de 4 núcleos. Também foi possível verificar que o incremento do número de processos acima do limite de 8 processos se torna inviável neste cenário, devido à limitação do grau de aproveitamento dos recursos de hardware disponíveis (*e.g.*, troca de contextos dos processos em execução, abertura e encerramento de conexões e troca de mensagens).

Além disso, foi realizada uma avaliação qualitativa, tendo em vista que o objetivo principal do *framework* proposto é viabilizar o processo de composição e o gerenciamento do ciclo de vida de SFCs através de uma interface que eleva o nível de abstração requerido. A Figura 3.6 ilustra um exemplo dos passos necessários para realizar a composição de uma *service chain* através do protótipo da aplicação cliente (*i.e.*, *REST Client*) executando em linha de comando. É importante lembrar que a aplicação cliente ilustrada neste exemplo apenas executa requisições REST ao componente *SFC-Core* que é responsável por implementar e abstrair a lógica para a definição das SFCs.

Inicialmente o usuário seleciona a opção "*7. Compose SFC*" com o intuito de definir uma nova SFC. A aplicação cliente envia uma mensagem ao *SFC-Core* solicitando a lista de

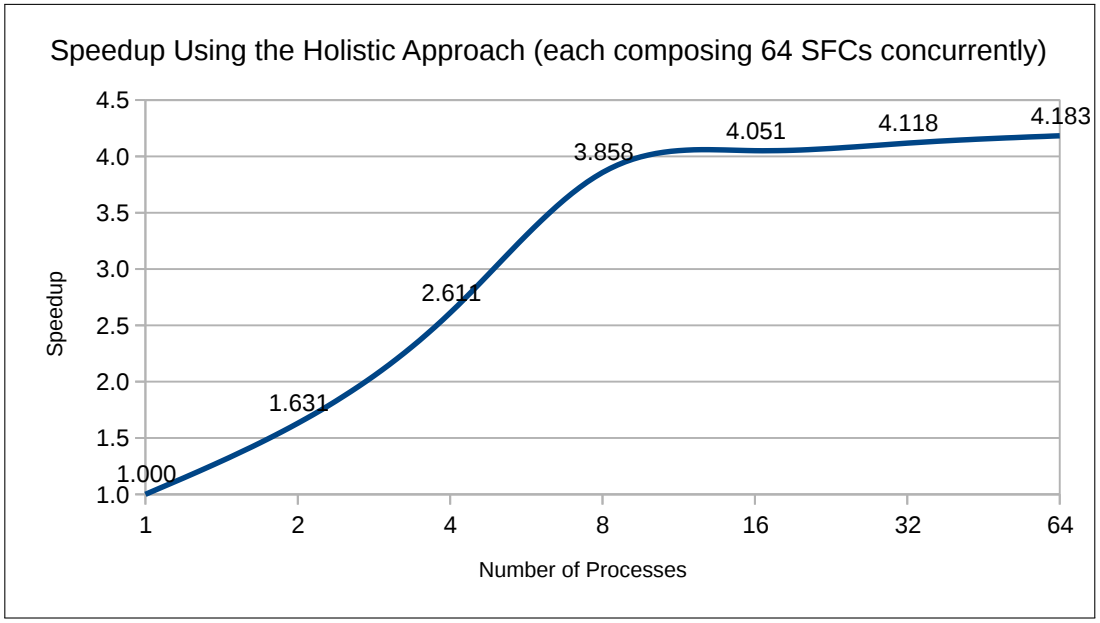


Figura 3.5: Gráfico de *speedup* do *framework* proposto.

todas as VNFs disponíveis no catálogo local e as apresenta na tela ao usuário. Em seguida, o usuário seleciona as VNFs "*1 - forwarder*" e "*2 - firewall*" que são encadeadas respectivamente na SFC pelo *SFC-Core*. Os componentes *SFC-Core* e *Tacker Agent* foram capazes de selecionar automaticamente os *Connection Points* de entrada e saída das VNFs, pois o usuário não necessitou informá-los na aplicação cliente.

Após o usuário encerrar o encadeamento das VNFs na SFC é necessário informar a origem do tráfego de entrada da SFC. O tráfego de entrada pode ser originado a partir de uma rede interna (*e.g.*, VNFs gerenciadas pelo Tacker) ou a partir de uma rede externa (*e.g.*, dispositivos não gerenciados pelo Tacker). Neste exemplo, depois de escolher a origem interna "*1 - Internal*" é apresentada a lista de VNFs instanciadas no orquestrador NFV. O usuário então seleciona a VNF "*2 - linux-client*" a qual será usada como o dispositivo de origem para o tráfego de entrada da SFC.

A configuração do tráfego de entrada da SFC neste caso foi realizada automaticamente, pois a VNF selecionada possui apenas um *Connection Point* de saída que é associado ao mesmo *Virtual Link* do *Connection Point* de entrada da primeira VNF da SFC. A lista de restrições (*i.e.*, ACL) é apresentada ao usuário depois de escolher o dispositivo de origem do tráfego da SFC. Após a configuração das restrições da ACL, o componente *SFC-Core* submete e instancia os descritores da SFC (*i.e.*, VNFFGD) e das VNFs (*i.e.*, VNFD) ao orquestrador NFV OpenStack Tacker por meio do agente de comunicação.

Com base no exemplo ilustrado na Figura 3.6 é possível constatar que a aplicação cliente executando por CLI permitiu guiar o usuário durante a tarefa de composição da SFC. A aplicação cliente realizou a composição de uma *service chain* consumindo apenas a API REST provida pelo componente *SFC-Core*. O *SFC-Core* foi capaz de compor e instanciar a SFC e suas VNFs programadas com Click-on-OSv no OpenStack Tacker utilizando apenas a API REST consumida pelo respectivo agente de comunicação. Logo, para adicionar outras plataformas NFV na arquitetura proposta necessita-se apenas codificar os respectivos agentes de comunicação. Também é possível observar que a abordagem do *framework* proposto permitiu a composição da SFC informando apenas a sequência de VNFs participantes da SFC. Isto permitiu simplificar as operações necessárias para compor SFCs e abstraiu as particularidades do orquestrador NFV

```
1. Include VNF Package
2. Remove VNF Package
3. Show VNF Catalog
4. Instantiate VNF
5. Destroy VNF Instance
6. Show VNF Instances
7. Compose SFC
8. Destroy SFC
9. Show SFC Instances
0. Exit

> 7
+-----+---------------+-----------------------------------+
| SEQ |  VNF Category |            Description            |
+-----+---------------+-----------------------------------+
|  1  |   forwarder   |       Click-on-OSv forwarder      |
|  2  |    firewall   |       Click-on-OSv firewall       |
|  3  | load_balancer |     Click-on-OSv Load Balancer    |
|  4  |      DPI      | Click-on-OSv Deep Packet Inspection |
|  5  |     linux     |           Linux Client            |
|  6  |  http_server  |         Linux HTTP Server         |
+-----+---------------+-----------------------------------+

Add VNFs by their Catalog SEQ (0 for done, -1 to exit)
SEQ > 1
VNF included successfully!
SEQ > 2
VNF included successfully!
SEQ > 0
Incoming SFC Network Traffic?
1. Internal
2. External
> 1
+-----+----------+--------------+----------------+--------+
| SEQ | VNF Name | Instance Name |  Mgmt Address  | Status |
+-----+----------+--------------+----------------+--------+
|  1  | 6GK98UA5 |  http-server  | 192.168.120.9  | ACTIVE |
|  2  | 3YIF1UXP |  linux-client | 192.168.120.13 | ACTIVE |
+-----+----------+--------------+----------------+--------+
Choose a VNF that generates the incoming SFC traffic (-1 to exit)
SEQ > 2
+------------------------+---------------------------------------+
|           ID           |              Description              |
+------------------------+---------------------------------------+
|         arp_op         |               ARP opcode              |
|        arp_sha         |      ARP source hardware address      |
|        arp_spa         |        ARP source ipv4 address        |
|        arp_tha         |      ARP target hardware address      |
|        arp_tpa         |        ARP target ipv4 address        |
| destination_port_range |           Target port range           |
|        eth_dst         |      Ethernet destination address     |
|        eth_src         |        Ethernet source address        |
|        eth_type        |  Ethernet frame type (See IEEE 802.3) |
|      icmpv4_code       |               ICMP code               |
|      icmpv4_type       |               ICMP type               |
|      icmpv6_code       |              ICMPv6 code              |
|      icmpv6_type       |              ICMPv6 type              |
|     ip_dst_prefix      |    IP destination address prefix      |
|        ip_proto        |          IP protocol number           |
|     ip_src_prefix      |       IP source address prefix        |
|        ipv6_dst        |       IPv6 destination address        |
|      ipv6_flabel       |           IPv6 Flow Label             |
|        ipv6_src        |         IPv6 source address           |
|       mpls_label       |               MPLS Label              |
|        vlan_id         |                VLAN ID                |
+------------------------+---------------------------------------+

Add the criteria by their respective ID (0 for done, -1 to exit)
ID > ip_proto
Value > 6
ID > destination_port_range
Value > 80-80
ID > ip_dst_prefix
Value > 10.10.0.5/32
ID > 0

SFC instantiated successfully!
```

Figura 3.6: Exemplo de composição de uma SFC através do protótipo de uma aplicação cliente.

OpenStack Tacker. Embora o exemplo não tenha ilustrado, a implementação do *SFC-Core* também possibilita que múltiplas composições de SFCs sejam realizadas de forma paralela.

3.5 COMPARAÇÃO COM OUTRAS ABORDAGENS

A importância das tecnologias relacionadas à NFV está refletida nos diversos trabalhos realizados nesta área. Algumas dessas iniciativas buscam solucionar problemas referentes à composição de VNFs para formar serviços de rede por meio de *Service Function Chaining*.

O projeto OSM (*Open Source MANO*) (ETSI, 2020b) tem como objetivo a implementação do *framework* NFV-MANO. Basicamente o OSM está dividido em três componentes principais: *openvim*, *openmano* e *openmano-gui*. O componente *openmano* é a referência para o bloco funcional NFVO do NFV-MANO, utiliza o modelo REST para disponibilizar funcionalidades de gerenciamento de VNFs, serviços de rede e os respectivos *templates*. O componente *openmano-gui* disponibiliza uma interface gráfica para o usuário modelar os descritores de VNFs e serviços de rede. O *openvim* é a implementação de referência para o bloco funcional VIM do *framework* NFV-MANO e faz a interface com os nodos computacionais da infraestrutura NFV, além de utilizar um controlador OpenFlow para prover as funcionalidades da rede por meio de SDN.

Embora o projeto OSM ofereça interfaces para interagir com o *openvim* e o *openmano* através de linha de comando e uma interface gráfica para modelar a composição dos serviços, seus usuários necessitam lidar com a complexidade de definir a maioria das configurações dos descritores manualmente, o que implica na necessidade de conhecer as particularidades da plataforma. O OSM também não permite realizar a composição e execução de *service chains* formadas por VNFs com Click-on-OSv, pois não oferece suporte às operações de FCAPS requeridas por estas VNFs.

O Open Baton (Fokus e Tu, 2020) é um projeto de código aberto que também oferece a implementação da especificação NFV-MANO da ETSI. Os principais componentes do Open Baton incluem um *NFV Orchestrator* e um *VNF Manager* genérico para gerenciar o ciclo de vida das VNFs de acordo com os seus respectivos descritores. Também é oferecido um SDK (*Software Development Kit*) com um conjunto de bibliotecas que podem ser usadas para implementar e adicionar VNFMs específicos em sua arquitetura. O componente NFVO foi desenvolvido na linguagem Java utilizando o *framework spring.io* e permite a interconexão entre o NFVO e os diferentes VNFMs por meio de uma API REST. O NFVO utiliza o OpenStack como a implementação padrão para o componente VIM da especificação NFV-MANO e permite o registro de múltiplos pontos de presença.

Embora o projeto do Open Baton ofereça uma interface gráfica para gerenciar as VNFs e os serviços de rede, o gerenciamento das funcionalidades FCAPS é limitado à execução de *scripts* na máquina virtual que executa a VNF. O Click-on-OSv executa sobre o sistema operacional *unikernel* OSv. Logo, a plataforma Open Baton não suporta a execução das funcionalidades FCAPS do Click-on-OSv devido a necessidade de executar um processo adicional no OSv para a troca de mensagens entre o VNFM e o EM.

Em Salsano et al. (2017) é proposto o *framework* RDCL 3D (*Reusable Functional Blocks Description and Composition Language Design, Deploy and Direct*). Este *framework* permite editar e validar descritores de serviços e seus componentes, e, implantar serviços de rede virtualizados com VNFs executando sobre a plataforma ClickOS. O ClickOS (Martins et al., 2014) também executa as funções de rede baseadas no *Click Modular Router*, porém sobre o *unikernel* Mini-OS. Embora o Mini-OS permita obter alto desempenho no processamento do tráfego da rede, o Mini-OS somente é capaz de executar sobre o hipervisor Xen. O gerenciamento

das funcionalidades FCAPS do ClickOS deve ser realizada localmente, pois não oferece uma API que permita o acesso externo.

Embora o *framework* proposto por Salsano et al. (2017) também possibilite a composição de VNFs em diferentes plataformas NFV através de *plugins*, existe a necessidade da modificação do *openvim* para executar o ClickOS, enquanto que na proposta do *Holistic-Composer* não é necessário modificar o OpenStack para a execução do Click-on-OSv. Outra distinção entre as duas propostas refere-se à execução do *Click Modular Router*, enquanto que Salsano et al. (2017) optaram por utilizar o ClickOS que executa sobre o Xen Server, a proposta deste trabalho aposta no Click-on-OSv que é portável e pode ser executado em múltiplos hipervisores como o KVM, Xen Server, VMWare e VirtualBox.

O *framework* RDCL 3D utiliza descritores proprietários para descrever seus projetos de SFC e implementa a lógica para a composição e validação de *service chains* no lado cliente (*i.e.,* GUI). Na proposta do *Holistic-Composer*, o componente *SFC-Core* abstrai toda a complexidade da implementação e validação do processo de composição de *service chains* no lado servidor através de uma API REST. Isto propicia a integração desta proposta com uma quantidade superior de facilitadores NFV que não suportam funcionalidades para compor SFCs.

3.6 CONCLUSÃO

Este capítulo apresentou o *framework Holistic-Composer* que visa simplificar a composição e o gerenciamento do ciclo de vida de uma SFC que pode ser orquestrada por diferentes plataformas NFV. O *framework* é proposto como uma solução genérica e emprega os padrões do NFV-MANO da ETSI. A implementação do *SFC-Core* permitiu disponibilizar uma API REST única fornecendo operações genéricas para compor e gerenciar o ciclo de vida de SFCs. A composição de 128 SFCs de forma paralela foi 4,145 vezes mais rápida se comparada com a abordagem sequencial. Também foi possível compor uma SFC apenas com a informação da sequência de VNFs da composição abstraindo as particularidades do orquestrador NFV utilizado. Em seguida, o Capítulo 4 descreve a proposta da Multi-SFC que visa permitir a composição e execução de SFCs em múltiplos domínios orquestrados por diferentes plataformas NFV.

4 MULTI-SFC: COMPOSIÇÃO DE SERVIÇOS DE REDE SOBRE MÚLTIPLOS DOMÍNIOS E ORQUESTRADORES NFV

Este capítulo descreve uma abordagem que permite realizar a composição e execução de serviços de rede em múltiplos domínios federados e orquestrados por diferentes plataformas NFV: a Multi-SFC. Primeiramente, é apresentada a arquitetura da Multi-SFC. Em seguida, são descritos os detalhes de implementação e avaliação da Multi-SFC. Por fim, é apresentada uma comparação com outras abordagens de orquestração de serviços de rede.

4.1 A ARQUITETURA MULTI-SFC

Esta seção descreve a funcionalidade e a arquitetura da Multi-SFC, uma abordagem para a composição e execução de SFCs distribuídas em múltiplas nuvens, orquestradores NFV e domínios, participantes ou não de uma federação. Para descrever a funcionalidade, são apresentadas as operações fornecidas pela API da Multi-SFC. Destaca-se que, embora algumas soluções existentes permitam a execução de serviços de rede em múltiplos domínios, as mesmas apresentam fortes limitações em termos de interoperabilidade entre diferentes plataformas e orquestradores NFV. Desta forma, a Multi-SFC é a primeira abordagem que abre caminho para a construção flexível de SFCs sobre múltiplas nuvens, múltiplos domínios **e** múltiplos orquestradores NFV.

Assume-se que um domínio é formado por um conjunto de sistemas e redes operados por uma única organização ou autoridade administrativa (Quittek et al., 2014). Uma nuvem compreende um modelo que permite o acesso sob demanda a recursos de computação que podem ser rapidamente configurados e provisionados/liberados com esforços mínimos de gerenciamento (Mell e Grance, 2011). Uma ou mais nuvens são hospedadas em cada domínio. Cada nuvem executa uma plataforma NFV que corresponde a um conjunto de sistemas que implementa a arquitetura NFV-MANO.

Além disso, uma federação consiste de um grupo de provedores de identidade (IdP - *Identity Provider*) e provedores de serviços (SP - *Service Providers*) que operam em uma relação de confiança mútua e se comunicam através de canais de comunicação seguros. O objetivo de uma federação é permitir que os usuários de um determinado domínio sejam capazes de acessar recursos e serviços de outros domínios com segurança e de forma integrada através de identidades federadas. A relação de confiança da federação é estabelecida através de acordos comerciais e permite formar uma aliança de identidades que são compartilhadas por meio de um protocolo comum entre todos os parceiros envolvidos (Laurent e Bouzefrane, 2015; Keltoum e Samia, 2017).

A Multi-SFC viabiliza a composição de serviços complexos de rede formados por VNFs que executam em diferentes nuvens e domínios, federados ou não. O bloco básico de construção de uma Multi-SFC é o *segmento* que consiste de um conjunto de VNFs que executam em uma única nuvem/domínio/plataforma. A Figura 4.1 apresenta o conceito de segmentação da Multi-SFC. Cada par de segmentos diferentes é interconectado por meio de um túnel VNF. Os túneis são baseados em tecnologias como VPN e VXLAN, e, instanciados como VNFs nos pontos de entrada e saída dos segmentos SFC que estão sendo conectados. Após um segmento ser instanciado em um domínio específico, o mesmo é conectado aos segmentos em execução de outros domínios que podem fazer parte de uma federação. Na medida em que há grande

quantidade de plataformas NFV disponíveis, diferentes domínios podem compor seus segmentos utilizando diferentes tecnologias e plataformas NFV.

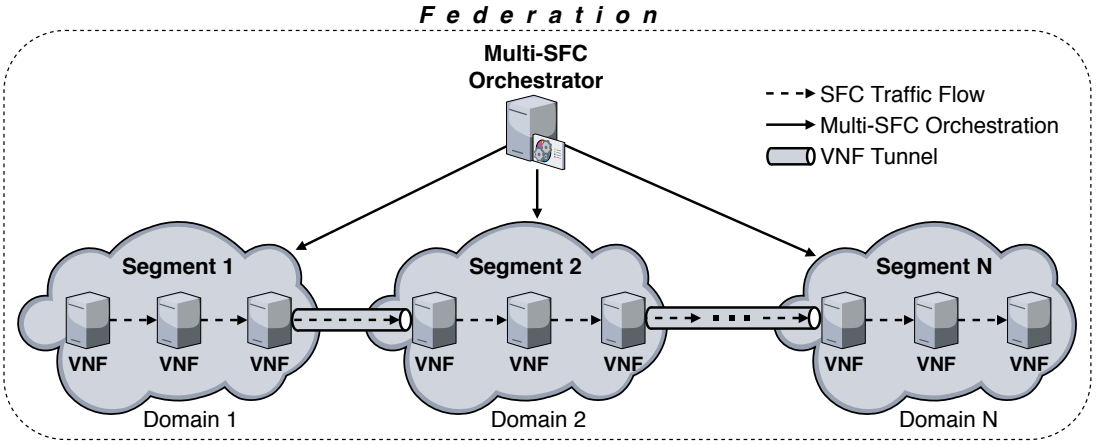


Figura 4.1: Multi-SFC: Segmentação.

O *Multi-SFC Orchestrator* mostrado na Figura 4.1 é responsável por gerenciar o ciclo de vida da Multi-SFC que consiste na composição, instanciação, execução e destruição dos segmentos que são distribuídos em múltiplos domínios/nuvens/plataformas. O *Multi-SFC Orchestrator* abstrai a composição e o gerenciamento das Multi-SFCs por meio de uma API genérica e de alto nível. Esta API possibilita que o operador de rede especifique a composição da Multi-SFC como uma sequência de VNFs e seja capaz de mapear a nuvem/domínio/plataforma em que cada VNF será instanciada.

A Figura 4.2 ilustra a arquitetura de uma Multi-SFC composta por dois domínios (*Domain 1* e *Domain 2*) que fazem parte da mesma federação (*Federation*). Inicialmente a SFC é composta por 6 VNFs que estão distribuídas entre dois segmentos divididos em um par de domínios. Túneis implementados como VNFs são empregados para conectar os segmentos da Multi-SFC nos múltiplos domínios e tornam-se parte da composição da Multi-SFC. Esta estratégia permite uma comunicação transparente entre os domínios com uma VNF implementando um túnel conectado ao final de um segmento (*VNF7*) e outra VNF implementando o mesmo túnel conectada ao início do próximo segmento (*VNF8*). Os túneis que interconectam os diferentes domínios executam as funcionalidades de SFF (*Service Function Forwarder*) conforme a nomenclatura da SFC da IETF (Halpern e Pignataro, 2015). O *Multi-SFC Orchestrator* realiza a configuração de rede necessária para encaminhar o tráfego através desses segmentos.

A Figura 4.3 apresenta a arquitetura do *Multi-SFC Orchestrator* que é proposta como uma solução genérica e extensível alinhada às especificações da arquitetura NFV-MANO. O NFV-MANO também é mostrado na figura para ilustrar a sua interação com os módulos do *Multi-SFC Orchestrator*. A arquitetura proposta permite a integração de diferentes plataformas NFV em múltiplos domínios e executando múltiplas aplicações de usuário.

O *Multi-SFC Core* é o módulo principal do *Multi-SFC Orchestrator*. Este módulo é responsável por coordenar a composição das SFCs em múltiplos orquestradores NFV, gerenciar os túneis da Multi-SFC, bem como validar as requisições executadas pelas aplicações cliente. O *Multi-SFC Core* utiliza um EM (*Element Management*) implementado para realizar a configuração dos túneis VNF entre os domínios. De acordo com a arquitetura NFV da ETSI (Quittek et al., 2014), o EM é responsável por realizar as operações FCAPS (*Fault, Configuration, Accounting, Performance, and Security*) das VNFs. Uma API de comunicação única e genérica é disponibilizada pelo *Multi-SFC Core* e fornece uma interface REST para a composição e o gerenciamento das Multi-SFCs. Assim, as várias aplicações cliente com arquiteturas diversas

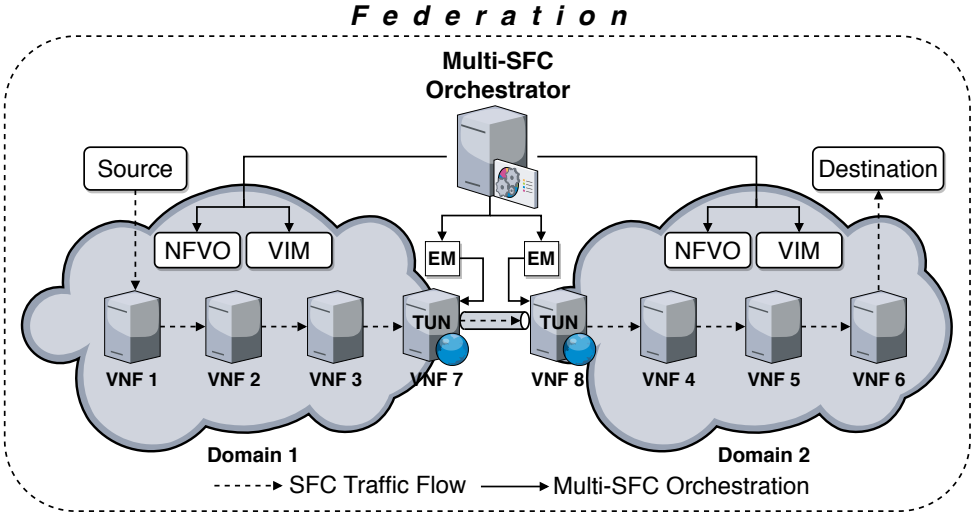


Figura 4.2: Arquitetura de uma Multi-SFC entre pares de domínios em uma federação.

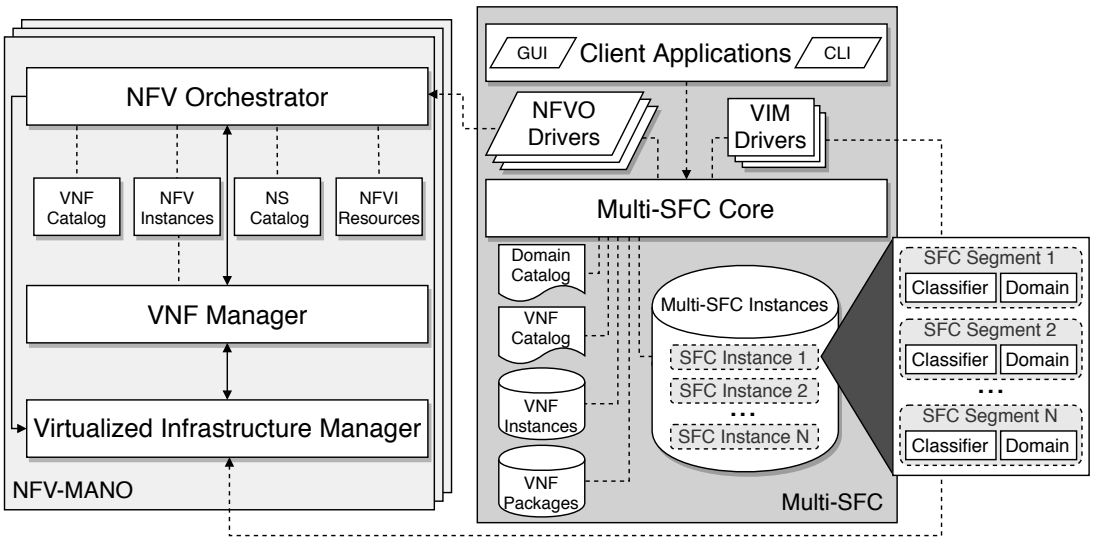


Figura 4.3: Arquitetura do *Multi-SFC Orchestrator*.

implementadas em CLI, GUI e outros sistemas podem fazer o uso da Multi-SFC para realizar a composição e gerenciamento de SFCs que atravessam em múltiplos domínios orquestrados por diferentes plataformas NFV. Uma visão geral das operações providas pela API do *Multi-SFC Core* para as aplicações cliente (*Client Applications*) é apresentada da Tabela 4.1.

As operações providas pela API do *Multi-SFC Core* às aplicações cliente para compor uma Multi-SFC são descritas a seguir:

- `GET /msfc/uuid`: gera e recupera um identificador único (`uuid`) para compor uma nova Multi-SFC (`msfc`). O identificador único é utilizado em várias operações para especificar e recuperar informações dos diferentes segmentos que formam uma mesma Multi-SFC.

- `GET /catalog/domains`: retorna as informações de todos os domínios armazenados no *Domain Catalog*. Para cada domínio são retornadas informações de todos os NFVOs, VIMs e tecnologias de túnel VNF disponíveis naquele domínio. Esta operação é usada para consultar se um par de domínios corresponde em termos de tecnologias de

Tabela 4.1: Visão geral das operações da API do *Multi-SFC Core*.

Operação	Descrição
`GET /msfc/uuid`	recupera um identificador único para compor uma nova Multi-SFC
`GET /catalog/domains`	recupera informações de todos os domínios armazenados no catálogo de domínios
`GET /catalog/vnfs/<dom-id>/<plat-id>`	lista todos os pacotes de VNFs de um determinado domínio/plataforma armazenados no catálogo de VNFs
`POST /msfc/sfp/compose`	adiciona uma VNF na composição de um segmento
`POST /msfc/source`	configura a origem do tráfego de entrada da Multi-SFC (*i.e.*, interna ou externa)
`GET /msfc/acl/<sfc-id>`	retorna todas as políticas suportadas pelo classificador da plataforma NFV do primeiro segmento da Multi-SFC
`POST /msfc/acl`	configura os classificadores de todos os segmentos ao longo da Multi-SFC
`GET /tunnel/em`	retorna a implementação do EM usado para configurar os túneis VNF
`POST /msfc/start`	instancia todos os segmentos de uma Multi-SFC nos diferentes domínios

túnel disponíveis e também para coletar informações sobre os endereços (*i.e.*, *endpoints*) e métodos de autenticação disponíveis em cada plataforma NFV.

- `GET /catalog/vnfs/<dom-id>/<plat-id>`: lista todos os pacotes de VNFs armazenados no catálogo de VNFs *VNF Catalog* que pertencem a um determinado domínio (`<dom-id>`). Visto que múltiplas plataformas NFV são permitidas e os pacotes de VNFs podem ter restrições ou requisitos específicos para executar em uma plataforma NFV em particular, apenas os pacotes de VNF compatíveis (`<plat-id>`) são retornados por esta operação. As restrições de execução são configuradas ao submeter o pacote VNF para o repositório *VNF Packages*.

- `POST /msfc/sfp/compose`: realiza a composição de um segmento por meio do encadeamento de VNFs. Esta operação recebe como entrada o domínio, o segmento e o identificador do pacote da VNF que está armazenado no *VNF Catalog*. Uma única VNF é encadeada em cada requisição desta operação. Além disso, a interface de entrada da rede da VNF sendo encadeada é conectada com a interface de saída da rede da VNF anterior já encadeada. Uma lista de interfaces é retornada para o usuário caso a interface de saída da VNF que está sendo encadeada não possa ser configurada automaticamente. A interface de saída da rede é escolhida com base nas seguintes regras: se houver apenas uma interface de rede, a interface de saída é a mesma que a interface de entrada; se houver mais interfaces de rede (exceto uma interface de gerenciamento), então uma lista com as interfaces disponíveis é retornada à aplicação cliente para que o usuário escolha a interface de saída desejada. A interface de saída selecionada é submetida para esta mesma operação que finaliza o encadeamento da VNF na SFC. Esta estratégia permite incluir VNFs que adicionam ramificações (*i.e.*, *VNF Branching*) na SFC e possibilita que o tráfego seja enviado para diferentes SFPs (*Service Function Paths*). Analisadores de Tráfego, Sistemas de Detecção de Intrusão e Balanceadores de Carga são exemplos de VNFs que podem executar em ramificações de uma SFC.

- `POST /msfc/source`: a configuração do tráfego de entrada da Multi-SFC é realizada após o encadeamento de todas as VNFs nos seus respectivos segmentos. A entrada desta operação indica se a origem do tráfego da rede do primeiro segmento da Multi-SFC é interna ou externa. O tráfego interno é gerado a partir de VNFs que executam e são gerenciadas pela própria plataforma NFV, enquanto que o tráfego externo é gerado a partir de um elemento de rede que está operando em uma rede não gerenciada pela plataforma NFV. Para o tráfego interno, o *Multi-SFC Core* permite ao usuário escolher se a origem do tráfego será de uma VNF em execução ou de uma VNF que será instanciada a partir do *VNF Catalog*. Para o tráfego externo, o *Multi-SFC Core* configura as políticas de *firewall* e roteadores virtuais para permitir que o tráfego de entrada seja encaminhado para o primeiro segmento da Multi-SFC.

- `GET /msfc/acl/<sfc-id>`: retorna as políticas suportadas pelo classificador da plataforma NFV que executa o primeiro segmento da Multi-SFC. As políticas de classificação do tráfego de uma SFC são definidas por meio de uma *Access Control List* (`acl`) que especifica atributos do tráfego de entrada como endereços de origem e destino, portas, protocolos, rótulos de fluxo, etc.

- `POST /msfc/acl`: recebe como entrada as políticas especificadas pelo operador de rede para o tráfego de entrada da SFC e configura os respectivos classificadores de cada plataforma NFV utilizada ao longo da Multi-SFC. As políticas de classificação dos segmentos são mapeadas e configuradas conforme a configuração do classificador do primeiro segmento e as respectivas plataformas NFV de cada segmento.

- `GET /tunnel/em`: retorna a implementação do EM usado para configurar os túneis VNF. Após a VNF que deverá executar o túnel ser instanciada, ela obtém o EM correspondente para realizar a autoconfiguração. Isso permite que o EM seja atualizado sem a necessidade de modificar as imagens das VNFs que implementam os túneis.

- `POST /msfc/start`: instancia todos os segmentos de uma Multi-SFC em seus respectivos domínios e orquestradores NFV conforme o identificador da Multi-SFC recebido como argumento. Esta operação executa as tarefas necessárias para instanciar e configurar a Multi-SFC que incluem enviar pacotes de VNFs e descritores da SFC para as respectivas plataformas NFV, instanciar as VNFs em seus respectivos segmentos, configurar os túneis VNF, configurar o roteamento entre os segmentos e configurar as políticas de segurança em cada plataforma NFV. Esta operação executa efetivamente a instanciação e configuração de toda a Multi-SFC após as operações anteriores terem configurado com sucesso os descritores de serviços de rede de cada segmento.

A API do *Multi-SFC Core* descrita anteriormente não é exaustiva. Outras operações também incluem administrar pacotes de VNFs, gerenciar descritores de VNFs (*i.e.*, VNFDs) nas plataformas NFV, bem como operações para instanciar, acessar e encerrar VNFs em múltiplas nuvens/domínios/plataformas.

A Figura 4.3 também apresenta os módulos *NFVO Drivers* e *VIM Drivers*. O módulo *NFVO Drivers* é responsável pela abstração da comunicação com diferentes orquestradores NFV que podem ser utilizados pelo *Multi-SFC Core*. As operações genéricas do *Multi-SFC Core* são traduzidas pelo módulo *NFVO Drivers* para as operações e particularidades de seus respectivos orquestradores NFV. Cada *NFVO Driver* implementa um conjunto de funcionalidades que permitem a composição e orquestração de SFCs que incluem desde o gerenciamento de VNFs e descritores das SFCs até a instanciação e destruição das mesmas.

As operações que um *NFVO Driver* deve incluir estão relacionadas à instanciação, monitoramento e destruição de VNFs e SFCs, bem como operações de gerenciamento dos diferentes SFPs das Multi-SFCs. As operações do módulo *NFVO Drivers* são descritas a seguir:

- `compose_sfp`: conecta VNFs de um SFP em um segmento específico da Multi-SFC usando as informações disponíveis nos respectivos *VNF Packages*.
- `get_sfc_traffic_src`: recupera uma lista de VNFs que podem ser configuradas para gerar o tráfego de entrada do primeiro segmento da Multi-SFC. Somente são elegíveis as VNFs ou *VNF Packages* que estão configurados para operar na mesma rede virtual do primeiro segmento da Multi-SFC.
- `configure_traffic_src_policy`: configura o classificador SFC para encapsular e encaminhar o tráfego de entrada – que pode ser interno ou externo – de cada segmento. Essa operação seleciona as interfaces de rede da infraestrutura de nuvem que serão usadas para encaminhar o tráfego interno (*VNF Packages* ou *VNF Instances*) e externo (roteadores virtuais) para a Multi-SFC.
- `get_available_policies`: retorna a lista de políticas e restrições que podem ser configuradas em um classificador de um determinado segmento da Muti-SFC. A lista corresponde à plataforma NFV que executará o respectivo segmento.
- `configure_policies`: configura os classificadores do tráfego de entrada de cada segmento da Multi-SFC. Esta operação é responsável por configurar todas as restrições do tráfego de entrada que foram definidas pelo usuário operador de rede da Multi-SFC. Uma Multi-SFC é especificada por um conjunto de descritores SFC – um para cada segmento/plataforma – que fazem parte de uma determinada composição.
- `get_configured_policies`: retorna a lista de políticas configuradas nos classificadores da Multi-SFC. O módulo *Multi-SFC Core* utiliza esta operação para configurar os classificadores dos segmentos Multi-SFC, túneis VNF e regras de *firewall* na infraestrutura NFV. A configuração das regras de *firewall* é necessária para autorizar o encaminhamento do tráfego da rede aos respectivos segmentos da Multi-SFC.
- `create_sfc`: implementa as respectivas operações em cada NFVO para instanciar um determinado segmento da Multi-SFC em uma determinada infraestrutura de nuvem.
- `destroy_sfc`: encerra um segmento da Multi-SFC e libera todos os recursos que foram reservados pelo orquestrador NFV durante a instanciação do mesmo.

A lista de operações requeridas pelo módulo *NFVO Drivers* descrita anteriormente não é exaustiva. Existem outras operações internas do módulo *NFVO Drivers* que implementam as particularidades de cada operação descrita acima.

O módulo *VIM Drivers*, também apresentado na Figura 4.3, é responsável pela configuração e interoperabilidade multi-domínio e multi-plataforma. Cada *VIM Driver* configura as respectivas redes, gerencia as regras de roteamento entre domínios e mantém as políticas de segurança necessárias para encaminhar o tráfego para cada segmento da Multi-SFC. Cada *VIM Driver* implementa as especificidades de um determinado VIM e deve estar à disposição do *Multi-SFC Orchestrator*. Uma lista não exaustiva com as principais operações que um *VIM Driver* deve implementar é descrita a seguir:

- `configure_networks`: verifica e configura os requisitos de rede para um determinado segmento Multi-SFC na infraestrutura de nuvem. Um exemplo é o endereçamento das sub-redes que não podem ser sobrepostas para evitar problemas na configuração de roteamento. Neste caso, o *VIM Driver* deve configurar um intervalo de sub-rede diferente.

- `configure_routes`: aplica todas as configurações de roteamento necessárias para direcionar o tráfego entre os segmentos Multi-SFC.

- `get_router_traffic_interface`: retorna a interface de rede do roteador a partir da qual o tráfego da rede externa será encaminhado para o classificador da Multi-SFC. Esta informação é usada pelo *NFVO Driver* para configurar as políticas do tráfego de entrada no classificador da Multi-SFC conforme descrito anteriormente.

- `configure_security_policies`: configura as restrições de tráfego de entrada para cada segmento da Multi-SFC em cada domínio. Esta operação configura todas as regras de *firewall* necessárias para permitir que o tráfego de entrada da Multi-SFC seja encaminhado para o classificador e o respectivo segmento.

O elemento *VNF Catalog*, ilustrado Figura 4.3, gerencia os metadados dos pacotes de VNFs que estão armazenados no repositório *VNF Packages*. O elemento *Domain Catalog* fornece metadados necessários para a comunicação entre domínios como endereços de rede, tipos de orquestradores NFV, VIMs disponíveis e informações de autenticação. O elemento *VNF Instances* armazena as diferentes instâncias de VNF que estão em execução nas múltiplas plataformas NFV. Por fim, o elemento *Multi-SFC Instances* armazena e mapeia as informações relacionadas a cada instância da Multi-SFC. Cada instância armazenada mantém informações sobre seus respectivos segmentos e suas VNFs, configuração dos classificadores de cada segmento da SFC e informações sobre os domínios e plataformas que hospedam os referidos segmentos. Isso permite identificar a plataforma NFV que um determinado segmento da Multi-SFC está executando e possibilita que todos os recursos da nuvem sejam liberados ao encerrar uma Multi-SFC.

4.2 IMPLEMENTAÇÃO DO PROTÓTIPO DA MULTI-SFC

Um protótipo da Multi-SFC foi implementado como prova de conceito (Huff et al., 2020). A implementação utiliza vários facilitadores NFV, em particular: OpenStack (OpenStack, 2020a), Tacker (Tacker, 2020) e OSM (ETSI, 2020b). A própria Multi-SFC também pode ser considerada um facilitador SFC, uma vez que abstrai a composição e o gerenciamento do ciclo de vida de segmentos de uma SFC distribuídos em diferentes nuvens e domínios. A plataforma OpenStack é empregada como o bloco funcional VIM da arquitetura NFV-MANO, enquanto que ambas as plataformas Tacker e o OSM implementam as funcionalidades de NFVO e VNFM. O protótipo da Multi-SFC foi implementado em linguagem Python. A API do *Multi-SFC Core* disponibiliza uma interface REST que foi implementada usando a biblioteca *Flask* da linguagem Python. O suporte à federação da Multi-SFC foi implementado utilizando o servidor HTTP Apache e o módulo Shibboleth *libapache2-mod-shib2*. O Shibboleth emprega a especificação SAML (*Security Assertion Markup Language*) (Lockhart et al., 2008) para as identidades federadas.

Dois *NFVO Drivers* foram implementados usando a biblioteca *Requests* da linguagem Python para trocar informações com as plataformas Tacker e OSM. Também foi implementado um *VIM Driver* para a plataforma OpenStack usando a biblioteca *Requests* da linguagem Python. Ambos os orquestradores (Tacker e OSM) utilizam o OpenStack para gerenciar o ciclo de vida

das VNFs e SFCs. Um EM foi implementado para gerenciar a configuração dos túneis VNF e permitir o encaminhamento do tráfego entre os segmentos da Multi-SFC. Foi utilizada a biblioteca *Flask* da linguagem Python para implementar o EM e disponibilizar uma API REST para o gerenciamento do ciclo de vida do túneis da Multi-SFC. O EM implementado é capaz de configurar túneis utilizando os protocolos IPSec, VXLAN e GRE. O gerenciamento do ciclo de vida dos túneis VNF foi implementado no módulo *Multi-SFC Core*. Cada túnel da Multi-SFC é estabelecido concretamente após todas as VNFs terem sido instanciadas em todos os segmentos da Multi-SFC.

A implementação da API do módulo *NFVO Drivers* abstrai a instanciação, consulta e encerramento dos segmentos da Multi-SFC distribuídos em diferentes domínios e orquestradores NFV. A implementação da API do módulo *VIM Drivers* abstrai a configuração de roteamento e as políticas de segurança necessárias para interconectar e encaminhar o tráfego da Multi-SFC para os segmentos nos diferentes domínios. Os classificadores dos segmentos da Multi-SFC são configurados com base nas políticas e restrições definidas pelo usuário operador de rede por meio de uma aplicação cliente que consome a API REST do *Multi-SFC Core*. A aplicação cliente foi implementada usando a biblioteca *requests-ecp* da linguagem Python que provê autenticação SAML. Essencialmente, a composição de uma Multi-SFC é transparente para o usuário que apenas necessita informar a sequência de VNFs de cada segmento e as conexões extras caso uma VNF possua múltiplas interfaces de rede. A composição das Multi-SFCs é baseada na abordagem holística descrita na Seção 3.3. O Apêndice A apresenta dois exemplos de composição de Multi-SFCs utilizando a aplicação cliente.

4.3 AVALIAÇÃO DO PROTÓTIPO IMPLEMENTADO

Os experimentos foram realizados em duas máquinas físicas e outras três máquinas virtuais executando sobre o sistema de virtualização KVM. Cada uma das máquinas físicas executa uma versão diferente do OpenStack e representam dois domínios diferentes. Estas máquinas executam as funcionalidades do bloco funcional VIM da arquitetura NFV-MANO da ETSI. A primeira máquina utilizada possui a seguinte configuração: CPU Intel(R) Core(TM) i7-6700HQ @ 2,6 GHz com 4 núcleos; 6144K de cache L3; 12 GiB de RAM; e, Linux Ubuntu 18.04. A segunda máquina possui a configuração: CPU Intel(R) Core(TM) i7-6700 @ 3,40 GHz com 4 núcleos; 8192K de cache L3; 8 GiB de RAM; e, Linux Ubuntu 18.04. As três máquinas virtuais executaram em outra máquina baseada no processador AMD Opteron(tm) 6136 @ 2,4 GHz com 24 núcleos; 96 GiB de RAM; e, Linux Ubuntu 20.04.

As plataformas Tacker e OSM desempenharam as funções dos blocos funcionais VNFM e NFVO nos experimentos realizados, cada um executando em uma máquina virtual independente. Uma terceira máquina virtual foi empregada como o controlador OpenStack para o OSM, enquanto que a máquina virtual do Tacker executou tanto o controlador Openstack como o NVFO e VNFM. Todas as máquinas virtuais foram configuradas com 8 vCPUs, 8 GiB de RAM e Ubuntu 18.04. As VNFs também são máquinas virtuais e cada uma delas foi configurada para executar com 1 vCPU, 256 MiB de RAM e Ubuntu Cloud 18.04. As máquinas físicas foram interconectadas por meio de uma rede Gigabit Ethernet.

A Figura 4.4 ilustra o cenário construído para avaliar a vazão e latência do tráfego da SFC utilizando a abordagem Multi-SFC. A ferramenta *iPerf3* foi utilizada para gerar tráfego e mensurar o *goodput* TCP entre o cliente *Client* de um domínio administrativo e os servidores *vServer* de outro domínio administrativo. Mensagens ICMP (*ping*) foram utilizadas para mensurar a latência entre a origem e o destino. A VNF *Firewall* foi implementada utilizando *iptables* e *iproute*. O *iptables* foi utilizado para filtrar e marcar os pacotes da rede e o *iproute* para encaminhar

os pacotes marcados para as diferentes interfaces de rede (*i.e.*, *VNF Branching*). Ambas as VNFs *DPI* (*Deep Packet Inspection*) neste experimento implementam apenas funcionalidades de encaminhamento de pacotes.

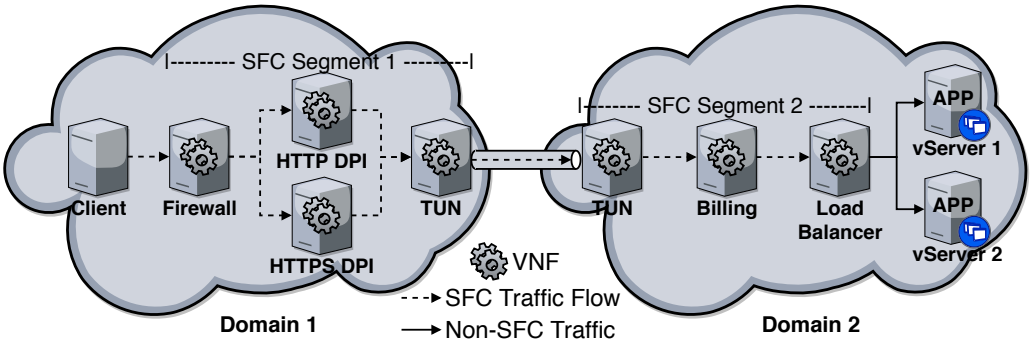


Figura 4.4: Cenário de avaliação da Multi-SFC.

Os túneis VNF executaram implementações concretas das tecnologias IPSec, VXLAN e GRE nos experimentos realizados. A VNF *Billing* também foi implementada com funcionalidades de encaminhamento de pacotes. A VNF *Load Balancer* foi implementada com *iptables* e distribui o tráfego para os servidores com base em um *hash* que é calculado a partir do endereço IP de origem. Assim, todas as conexões do mesmo cliente são entregues a um mesmo servidor. O encaminhamento do tráfego para um mesmo servidor é requisito da ferramenta *iPerf3* para mensurar o goodput TCP, uma vez que é necessário o estabelecimento de no mínimo duas conexões entre o cliente e o servidor *iPerf3*. Embora o cenário apresente dois *vServers*, todo o tráfego do experimento foi encaminhado para um mesmo servidor em cada execução do experimento.

O primeiro experimento analisa o impacto na vazão da rede entre o cliente (*Client*) e a VNF que implementa o túnel (*TUN*) no primeiro domínio (*Domain 1*). Também foi mensurada a vazão entre a VNF que implementa o túnel (*TUN*) e um *vServer* no segundo domínio (*Domain 2*). Este experimento serve de base para a comparação da vazão com os outros experimentos da Multi-SFC e foi executado sem encadear o tráfego na SFC. O experimento foi executado 30 vezes de 60 segundos cada. A média da vazão TCP no primeiro domínio foi próxima a 37 Gbps, enquanto que no segundo domínio a vazão média atingiu 26 Gbps. Como as VNFs *Client*, *vServer* e *TUN* executam sobre máquinas virtuais com as mesmas configurações, a diferença na vazão TCP dos dois domínios está relacionada à diferença nas CPUs das máquinas físicas.

Em seguida foi realizada a composição de dois segmentos SFC e o impacto da vazão TCP foi mensurada em cada um dos segmentos de forma independente. O tráfego da rede neste experimento também não foi encaminhado de um domínio para o outro. A vazão TCP no primeiro segmento (*SFC Segment 1*) foi em média 16,679 Gbps, enquanto que no segundo segmento (*SFC Segment 2*) a vazão atingiu em média 9,648 Gbps. A redução da vazão comparada com o experimento de base deve-se ao fato de que VNFs adicionais foram instanciadas nos respectivos segmentos em cada uma das infraestruturas de nuvem. Além disso, o tráfego da SFC deve ser encaminhado para e processado por cada uma das VNFs nos respectivos segmentos. Os classificadores de tráfego SFC de cada segmento também impõem uma sobrecarga extra. Ainda, o componente *Open vSwitch* (Linux Foundation, 2020b) utilizado pela plataforma OpenStack usa pares de interfaces *veth* (dispositivos Ethernet virtuais que se conectam através do *kernel*). Esses dispositivos são conhecidos por apresentar desempenho moderado (Panda et al., 2016) e por esta razão a vazão da SFC é inversamente proporcional à quantidade de VNFs encadeadas na SFC.

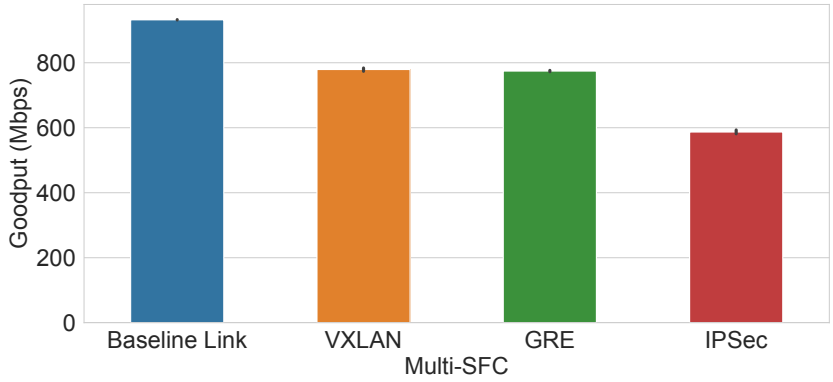


Figura 4.5: Vazão TCP entre domínios utilizando diferentes túneis VNF.

Também foi executado um experimento para avaliar o impacto da vazão TCP executando uma Multi-SFC completa que consiste nos dois segmentos mostrados na Figura 4.4. Este experimento forneceu medições desde a origem *Client* até o destino *vServer*. O tráfego foi encaminhado e processado pelas VNFs dos dois segmentos e também entre os dois domínios. Como o primeiro segmento possui uma ramificação *VNF Branching* para duas VNFs *DPI* seguida de uma interseção para o túnel VNF (*TUN*), o tráfego desse experimento foi processado por apenas uma das funções *DPI*. Os túneis VNF implementados para o experimento correspondem às tecnologias IPSec, VXLAN e GRE. A Figura 4.5 mostra a média de 30 execuções de 60 segundos cada, com intervalos de confiança de 99%.

Inicialmente foi mensurada a vazão entre as máquinas virtuais que implementam os túneis entre os dois domínios sem executar as outras VNFs, SFCs e o túnel IP. A vazão TCP deste experimento foi em média 932,1 Mbps. O resultado deste experimento deve-se ao fato de que as duas máquinas virtuais estão conectadas por meio de uma rede Gigabit Ethernet. A vazão entre as máquinas que executam o OpenStack também foi mensurada e obteve-se praticamente os mesmos resultados.

Também foi executado um experimento de uma Multi-SFC completa empregando o túnel VNF que implementa a VXLAN. A vazão média atingida neste experimento foi de 779,3 Mbps conforme mostrado na Figura 4.5. Este resultado já era esperado, pois o túnel VXLAN adiciona uma sobrecarga extra nos pacotes da rede (os tamanhos dos cabeçalhos dos pacotes são aumentados em 50 bytes cada). As duas VNFs que implementam os túneis IP também adicionam uma sobrecarga de processamento para encapsular e desencapsular o tráfego da VXLAN. A vazão de uma Multi-SFC completa executando um túnel VNF implementado com GRE também foi avaliada como alternativa ao túnel VXLAN. O encapsulamento GRE adiciona pelo menos 24 bytes extras por pacote. A Multi-SFC executando sobre o túnel GRE atingiu em média a vazão de 774,5 Mbps. Pode-se observar que os túneis GRE e VXLAN obtiveram aproximadamente o mesmo desempenho.

Ainda foi mensurada a vazão TCP de uma Multi-SFC completa empregando túneis VPN que implementam IPSec. A vazão média deste experimento atingiu 586,9 Mpbs. Neste caso, a redução da vazão TCP deve-se ao fato de que as VNFs que implementam o IPSec necessitam realizar chamadas de sistema para encriptar e decriptar o tráfego da rede (que é caro em termos computacionais). As tarefas de criptografia e encapsulamento do tráfego elevaram a percentagem de uso da CPU dos processos *qemu-system* do KVM próximos a 100% de utilização. Além disso, as VNFs foram configuradas apenas com uma CPU virtual. O hardware das máquinas físicas em que o OpenStack foi executado também pode ter influenciado na vazão do IPSec, uma vez que os quatro núcleos da CPU foram compartilhados entre as VNFs de cada segmento junto com os

processos necessários para executar o OpenStack. Embora uma das máquinas físicas tenha mais núcleos de processamento, ela não foi utilizada devido ao seu baixo desempenho para tarefas de Entrada/Saída (incluindo a rede).

Também foi executado um experimento para mensurar a latência da Multi-SFC. Este experimento avaliou a latência de uma Multi-SFC completa, na qual o tráfego foi encaminhado entre os dois domínios utilizando túneis VNFs implementando VXLAN, GRE e IPSec. Os resultados são mostrados na Figura 4.6. O experimento foi repetido por 30 vezes durante 60 segundos cada. As amostras ICMP foram coletadas a cada intervalo de 1 segundo.

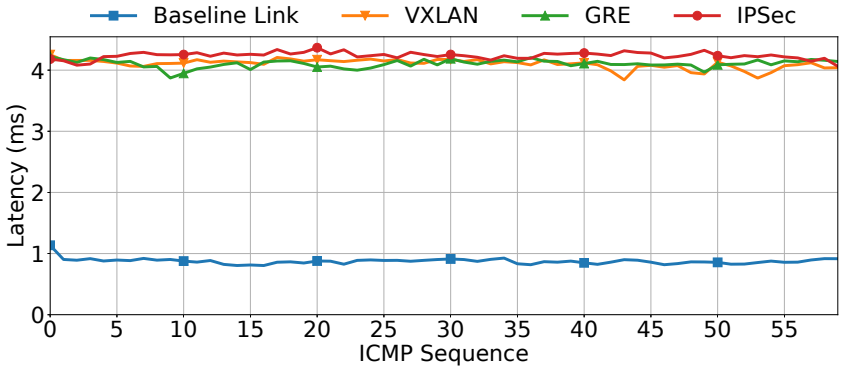


Figura 4.6: Latência mensurada utilizando diferentes túneis VNF.

Inicialmente foi mensurada a latência entre as máquinas virtuais que implementam os túneis entre os dois domínios sem executar as outras VNFs, SFCs e o túnel IP. A latência média foi 0,859ms com pouca variação nas diferentes rodadas de execução. A latência de uma Multi-SFC completa usando túneis VNF que implementam tanto VXLAN e GRE atingiu em média 4,1ms, enquanto que usando o túnel VNF implementado com IPSec a latência foi em média 4,23ms. A latência maior para o túnel IPSec também era esperada, uma vez que as VNFs do túnel necessitam tratar da criptografa dos pacotes e requerem mais ciclos de processamento para executar as suas tarefas. De modo geral, o aumento da latência foi causado por mensagens que são encaminhadas e também processadas por todas as VNFs e os túneis da Multi-SFC. Os classificadores também impõem uma certa latência, uma vez que as VNFs utilizadas desconhecem a existência da SFC (*SFC-unaware*) e, portanto, o tráfego necessitou ser reclassificado por um *SFC Proxy* após sair de cada uma das VNFs que compõem a SFC. O hardware com um número limitado de CPUs também influenciou na latência final da Multi-SFC.

Embora a Multi-SFC e a plataforma Openstack Tacker suportem a federação de domínios utilizando a especificação SAML, a versão utilizada da plataforma OSM (*Release Eight*) ainda não oferece suporte a identidades federadas. A API atual da plataforma OSM apenas suporta autenticação interna e controle de acesso baseado no modelo RBAC (*Role-based Access Control*) via Keystone (OpenStack, 2020b). Embora o Keystone suporte identidades federadas e é usado pelo Tacker e OpenStack, a plataforma OSM ainda não externaliza a API de autenticação e autorização através de federação. Por este motivo, os experimentos foram executados utilizando autenticação local em cada uma das plataformas.

4.4 COMPARAÇÃO COM OUTRAS ABORDAGENS

Diversas plataformas NFV implementam o bloco funcional NFVO da arquitetura NFV-MANO. Entretanto, nenhuma delas oferece suporte para a execução distribuída de SFCs sobre múltiplas nuvens e domínios orquestrados por diferentes plataformas NFV em uma federação. Os trabalhos a seguir propõem plataformas e *frameworks* para solucionar alguns desafios envolvidos.

Em Francescon et al. (2017) é proposto o *framework* X-MANO para a orquestração de serviços de rede em múltiplos domínios. De modo geral, o X-MANO somente permite acessar recursos e encadear serviços de rede pré-existentes e que foram disponibilizados pelas interfaces dos orquestradores NFV de cada domínio. Isso pode gerar desperdício de recursos computacionais e energia, uma vez que o tráfego da rede pode atravessar VNFs desnecessárias. Embora a arquitetura proposta para o X-MANO envolva a orquestração de serviços de rede que executam em múltiplos domínios com federação, em Baggio et al. (2017) os autores apresentam um estudo de caso simples realizado com a plataforma NFV Open Baton (Fokus e Tu, 2020). As VNFs que fazem a conexão inter-domínios deste estudo de caso devem ser instanciadas e configuradas manualmente pelos administradores de rede. Além disso, a configuração dos recursos disponíveis em cada domínio (*e.g.*, redes, VNFs e serviços de rede) é realizada manualmente no X-MANO, enquanto que na abordagem da Multi-SFC a disponibilização desses recursos é imediata/automática e podem ser instanciados em qualquer domínio com base na identidade e autorização do operador de rede na federação.

Um mecanismo para o estabelecimento automatizado de Redes Privadas Virtuais dinâmicas no contexto da NFV foi recentemente proposto (Gunleifsen et al., 2019). O objetivo é fornecer criptografia e segurança para conectar as funções de uma SFC. Embora Gunleifsen et al. (2019) propõem o estabelecimento automático de túneis VPNs, os autores assumem que cada domínio implementa a sua própria solução de composição e orquestração de SFCs multi-domínio. Além disso, a abordagem proposta pelos autores requer que as VNFs do túnel implementem o mesmo protocolo (*e.g.*, NSH, MPLS) usado pela SFC para rotear/encaminhar o tráfego para destino, enquanto que a abordagem da Multi-SFC é agnóstica. Na Multi-SFC, as VNFs que implementam o túnel fazem parte da SFC e por isso não necessitam processar o cabeçalho da SFC. O roteamento entre os segmentos é configurado com base nas informações do classificador da Multi-SFC. Assim, todo o tráfego com as mesmas políticas do classificador da Multi-SFC é encapsulado e encaminhado para o túnel e consequentemente para próximo segmento.

Em Salsano et al. (2017), é proposto o *framework* RDCL 3D (*Reusable Functional Blocks Description and Composition Language Design, Deploy and Direct*). O RDCL3D permite editar e validar descritores de serviços, bem como implantar SFCs compostas por VNFs executando sobre a plataforma ClickOS (Martins et al., 2014). Embora o RDCL 3D possibilite a composição de SFCs em diferentes plataformas NFV, são necessárias modificações no VIM (*Virtualized Infrastructure Manager*) (*e.g.*, openvim) para executar o ClickOS. Além de não abordar o contexto de múltiplos domínios, o RDCL 3D também emprega descritores proprietários para definir as SFCs e implementa a lógica para a composição e validação das SFCs no lado cliente.

O orquestrador NFV TeNOR (Riera et al., 2016) permite o gerenciamento de serviços de rede e o controle do ciclo de vida de VNFs sobre infraestruturas virtualizadas distribuídas. O orquestrador TeNOR automatiza a instanciação de SFCs e gerencia os recursos computacionais das VNFs que compõem a SFC. Embora uma solução baseada em mapeamento seja fornecida para instanciar uma SFC em múltiplos PoPs (Pontos de Presença), o TeNOR não permite criar SFCs em vários orquestradores NFV diferentes e também não suporta federação de domínios.

Open Platform for NFV (OPNFV) (Linux Foundation, 2020c) é um projeto NFV da Linux Foundation que desenvolve uma plataforma para simplificar e integrar o desenvolvimento e implantação de componentes NFV. O OPNFV fornece uma plataforma NFV de referência que permite analisar e testar a interoperabilidade entre soluções NFV e SDN de diferentes desenvolvedores. No entanto, não é possível compor e executar uma SFC distribuída em múltiplos domínios orquestrados por diferentes plataformas NFV, uma vez que o OPNFV não aborda a integração de diferentes orquestradores NFV.

O Blue Planet MDSO (*Multi-Domain Service Orchestration*) (Ciena, 2020) propõe um *framework* para o gerenciamento e composição de serviços de rede entre múltiplos domínios e infraestruturas NFV. Apesar de facilitar a configuração e instanciação de SFCs, o *framework* não permite a composição e orquestração de uma SFC que atravessa múltiplos domínios com diferentes plataformas de orquestração NFV.

Em (Sonkoly et al., 2015), os autores empregam a tecnologia SDN para distribuir SFCs em múltiplas nuvens e domínios administrativos. O sistema permite interconectar diferentes VIMs e acessar orquestradores de recursos da infraestrutura, mas não permite executar uma SFC composta por múltiplos diferentes orquestradores de serviços NFV. Além disso, o sistema depende estritamente de enlaces SDN para interconectar os múltiplos domínios e não considera federação.

Cloudify (Cloudify, 2020) é um projeto que permite a integração de funções de rede virtuais e físicas e fornece um NFVO e VNFM genéricos para orquestrar múltiplas nuvens federadas. O sistema emprega roteadores virtuais pré-configurados para interconectar diferentes nuvens por meio de túneis. Embora o Cloudify seja capaz de orquestrar várias nuvens, ele não permite que SFCs sejam construídas em múltiplos e diferentes orquestradores NFV.

A plataforma ONAP (2020) permite a orquestração, gerenciamento e automação de serviços de rede através de uma implementação agnóstica de fornecedor. A arquitetura da plataforma ONAP é constituída por diversos subsistemas de software e de forma geral é dividida em dois *frameworks* arquiteturais: projeto e execução. O *framework* de projeto é responsável pela definição e programação da plataforma. O *framework* de execução tem a função de processar a lógica e automatizar as políticas de execução que foram programadas na fase de projeto. Projetar serviços de rede no ONAP basicamente envolve a definição de políticas que são executadas de acordo com o comportamento do serviço. Embora a plataforma ONAP implemente *frameworks* que são independentes de fornecedor, ela não possibilita a composição e o gerenciamento SFCs que atravessam múltiplos domínios orquestrados por diferentes plataformas NFV.

O projeto pSmart (Joshi e Kataoka, 2020) concentra-se na composição de SFCs em vários domínios para reduzir custos de utilização de recursos computacionais. O pSmart também visa reduzir os riscos de privacidade e segurança e implementa um processo de decisão baseado em aprendizagem para mapear as SFCs nos diferentes domínios. Os recursos utilizados pelo pSmart para construir SFCs devem ser previamente mapeados e abstraídos das interfaces de comunicação específicas de cada orquestrador de domínio. Os enlaces de interconexão entre os diferentes domínios também devem ser previamente estabelecidos à composição da SFC.

O projeto 5G Exchange (5GEx) (Sgambelluri et al., 2017) tem o objetivo de coordenar a alocação e o uso eficiente de recursos de computação, armazenamento e rede para implantar serviços em redes 5G. As informações do catálogo de serviços, a topologia das redes e os recursos computacionais são compartilhados entre os orquestradores 5GEx MDO (*Multi-Domain Orchestrator*) para realizar a implantação dos serviços de rede em múltiplos domínios. Embora seja possível implantar serviços em múltiplos domínios, o projeto 5GEx não possibilita utilizar múltiplos diferentes orquestradores NFV e também não aborda o contexto de federações.

Em Uniyal et al. (2020) é proposta a arquitetura 5GUK Exchange (5GUKEx) que tem o objetivo de prover a orquestração de serviços virtualizados para as redes 5G. O foco é prover uma solução flexível e leve para criar serviços e redes de múltiplos domínios. Assim como na Multi-SFC, o 5GUKEx introduz uma camada de orquestração multi-domínio e utiliza o serviço de orquestração local de operador de rede. Entretanto, o 5GUKEx realiza a composição de serviços de rede encadeando os serviços pré-existentes em cada domínio e que foram publicados através de um componente chamado NSB (*Network Service Broker*). Na Multi-SFC, a composição é totalmente independente de serviços pré-existentes, um operador de rede pode compor o

serviço a partir de suas próprias VNFs por toda a federação de domínios garantindo maior flexibilidade de escolha do provedor de recursos virtualizados. Além disso, o 5GUKEx apenas suporta o uso da plataforma de orquestração Open Source MANO (OSM) e do controlador SDN OpenDaylight (ODL), enquanto que a Multi-SFC é multi-plataforma e é independente de tecnologia de interconexão por utilizar pares de VNFs para implementar os túneis.

Por fim, a plataforma FENDE (Bondan et al., 2019) oferece um ecossistema para projetar, desenvolver, instanciar e executar serviços de rede baseados em NFV. O ecossistema FENDE também permite criar SFCs baseadas em VNFs que podem ser adquiridas através de um *Marketplace* NFV. A infraestrutura da plataforma FENDE é dividida em três domínios que fazem parte de uma mesma federação. Embora os usuários da plataforma FENDE possam instanciar VNFs e compor serviços de rede que executam nos diferentes domínios da federação, apenas é oferecido o suporte a um orquestrador NFV (Tacker). A Multi-SFC suporta a composição e execução de SFCs em múltiplos domínios federados e orquestrados por múltiplas plataformas NFV.

A Tabela 4.2 apresenta uma comparação qualitativa das abordagens NFV descritas anteriormente. As colunas da tabela de comparação denotam as propriedades comparadas enquanto que as linhas referem-se às abordagens NFV. O símbolo ● denota que a abordagem proposta possui suporte a uma determinada propriedade, enquanto que o símbolo ○ indica que a proposta não apresenta aquela propriedade. As propriedades comparadas na tabela são as seguintes:

- *Múltiplos VIMs*: indica que a abordagem possibilita a execução de VNFs e SFCs sobre diferentes VIMs;
- *Múltiplos Orquestradores*: indica se existe suporte à interoperabilidade ou integração entre diferentes orquestradores NFV. O símbolo ◐ indica que são suportados diferentes orquestradores, porém não é possível compor e executar uma SFC de forma integrada entre os múltiplos domínios e orquestradores NFV suportados;
- *Múltiplos Domínios*: indica que a abordagem possui suporte para a execução de VNFs e SFCs em múltiplos domínios;
- *Identidade Federada*: denota se a abordagem implementa federação de domínios; e,
- *Provisionamento Imediato*: indica se um determinado recurso (*e.g.*, VNF, SFC) pode ser provisionado/instanciado em uma plataforma de outro domínio da federação sem a necessidade da intervenção manual do operador de rede daquele domínio. O símbolo ◐ indica que a abordagem é dependente de projetos e plataformas adicionais para prover esta propriedade.

Vários avanços foram realizados no desenvolvimento de plataformas, *frameworks* e orquestradores NFV para permitir a execução de SFCs distribuídas sobre múltiplas nuvens/domínios/orquestradores. No entanto, a Tabela 4.2 mostra que, antes da Multi-SFC, até então nenhuma solução fornece todo o *framework* para a composição e execução de SFCs que atravessam múltiplas nuvens em domínios federados orquestrados por diferentes plataformas NFV. Enquanto a maioria das soluções aborda a execução de SFCs em múltiplos VIMs e domínios, essas são limitadas em termos de federação e integração de SFCs entre diferentes orquestradores NFV.

Tabela 4.2: Comparação qualitativa com outras abordagens NFV.

	Múltiplos VIMs	**Múltiplos Orquestradores**	**Múltiplos Domínios**	**Identidade Federada**	**Provisionamento Imediato**
X-MANO	●	○	●	●	○
Gunleifsen *et al.*	○	○	●	○	○
RDCL 3D	●	◐	○	○	○
TeNOR	●	○	●	○	○
OPNFV	●	◐	●	●	◐
MDSO	●	○	●	●	●
Sonkoly *et al.*	●	◐	●	○	○
Cloudify	●	○	●	●	●
ONAP	●	○	●	●	●
pSmart	●	●	●	○	○
5GEx	●	○	●	○	○
5GUKEx	●	○	●	○	○
FENDE	●	○	●	●	●
Multi-SFC	●	●	●	●	●

4.5 CONCLUSÃO

Este capítulo apresentou a proposta da Multi-SFC para a composição e execução de SFCs que atravessam múltiplas nuvens em domínios federados ou não, e orquestrados por diferentes plataformas NFV. A Multi-SFC emprega a abordagem holística que abstrai composição e o gerenciamento do ciclo de vida de SFCs. Os segmentos de uma Multi-SFC são interconectados por meio de túneis implementados como VNFs que fazem parte da composição da SFC. Um protótipo foi implementado como prova de conceito utilizando as plataformas Tacker, OSM e OpenStack. Resultados experimentais mostram que a Multi-SFC apresenta baixa latência e mantém vazão satisfatória mesmo executando em hardware limitado. A sobrecarga dos túneis VNF tem relação com o protocolo de comunicação utilizado e varia conforme o tipo de serviço oferecido (*e.g.*, criptografia). Também foi realizada uma comparação com outras abordagens NFV. Foi possível verificar que as soluções apresentadas são limitadas em termos de integração de orquestradores NFV para permitir o provisionamento e execução de SFCs que atravessam múltiplas nuvens e domínios federados. Embora esteja alinhada com as especificações NFV-MANO da ETSI, a Multi-SFC pode ser utilizada com outras implementações que não seguem o NFV-MANO. A premissa é que as implementações correspondentes dos módulos *VIM Drivers* e *NFVO Drivers* disponibilizem operações básicas do ciclo de vida de VNFs e SFCs (*i.e.*, composição, instanciação, execução e encerramento).

5 RFT: MICROSSERVIÇOS ESCALÁVEIS E TOLERANTES A FALHAS PARA O CONTROLADOR O-RAN

Todos os capítulos anteriores desta Tese de Doutorado tiveram como foco a tecnologia NFV. Este capítulo descreve uma contribuição que foi desenvolvida em outra área: RAN (*Radio Access Network*). Este trabalho começou a ser desenvolvido durante o estágio sanduíche realizado na AT&T Shannon Labs em New Jersey, USA. Apesar de que o objetivo inicial era continuar trabalhando com NFV, os planos foram alterados por motivos diversos. Assim, este capítulo descreve a RFT (*RIC Fault Tolerance*): uma estratégia que permite a implementação de microsserviços tolerantes a falhas para o controlador definido nos padrões O-RAN (*Open RAN*). No capítulo, inicialmente é apresentada uma contextualização geral da O-RAN e da replicação com a abordagem máquina de estados. Em seguida, é descrita uma visão geral da plataforma RIC e dos seus microsserviços que são chamados xApps. A biblioteca RFT para o desenvolvimento de xApps tolerantes a falhas é então descrita. As técnicas para a gerência do grupo de réplicas de um xApp são apresentadas. A replicação parcial de estados dos xApps utilizada pela RFT é descrita na sequência. Uma avaliação empírica é apresentada na sequência. Por fim, são apresentados os trabalhos relacionados. A API da RFT para a implementação de xApps tolerantes a falhas é apresentada no Apêndice B.

5.1 CONTEXTUALIZAÇÃO

A RAN (*Radio Access Network*) é parte fundamental das redes móveis, fornecendo conectividade para dispositivos sem fio dos mais diversos tipos, denominados UE (*User Equipment*). Embora as interfaces de comunicação entre os dispositivos sem fio e os elementos da RAN sejam definidas por padrões abertos, a maioria das implementações são normalmente soluções proprietárias fornecidas por fabricantes de equipamentos (Olwal et al., 2016; Parvez et al., 2018; Habibi et al., 2019). Recentemente, a O-RAN Alliance (2019) e a O-RAN SC (2020b) (*O-RAN Software Community*) vêm coordenando esforços para apoiar o desenvolvimento de software de código aberto para a RAN considerando requisitos de desempenho, escalabilidade e alinhamento com os padrões especificados pelo projeto 3GPP (*3rd Generation Partnership Project*) (3GPP, 2020a). Uma nova interface de comunicação chamada E2 está sendo definida para permitir que elementos da RAN sejam capazes de expor suas funcionalidades e reportar notificações ao controlador O-RAN. A O-RAN SC (2020b) vem liderando esforços para a implementação e disponibilização de um controlador O-RAN de código aberto denominado RIC (*RAN Intelligent Controller*). O RIC (O-RAN SC, 2020a) pode ser visto como uma plataforma para executar microsserviços, chamados xApps, que usam a interface E2 para acessar e controlar os elementos da RAN.

A interação entre os xApps e os elementos da RAN possui requisitos rígidos em termos de desempenho, em particular a latência (Coletti et al., 2018; Akman et al., 2020; O-RAN Alliance, 2020c,a). Para exemplificar, considere um xApp responsável por controlar a admissão de UEs em uma rede e fornecer proteção contra ataques de DDoS (*Distributed Denial of Service*) intencionais ou acidentais no contexto de IoT (Liang et al., 2016; Bhardwaj et al., 2018). Nesse cenário, a plataforma RIC e os xApps devem apresentar baixa latência para que as operações válidas não sejam atrasadas e a experiência do usuário não seja prejudicada. A tolerância a falhas também é um requisito importante, pois um xApp com falhas poderia impedir a conexão de UEs

legítimos na rede. Assim, é importante ter xApps tolerantes a falhas, mas devem assegurar o desempenho necessário.

A alta disponibilidade, que é um requisito fundamental para a RAN, pode ser alcançada com técnicas de replicação e tolerância a falhas. Uma estratégia para alcançar a tolerância a falhas é utilizar a abordagem de replicação de máquina de estado. Essa abordagem permite a implementação de um serviço tolerante a falhas através da replicação de diferentes instâncias daquele serviço. Uma máquina de estado consiste em variáveis que caracterizam o estado e de comandos, que executados modificam o estado. Cada comando é implementado por um processo determinístico executado de forma atômica com relação aos outros comandos, produz uma saída e modifica o estado atual. Clientes da máquina de estado realizam requisições para executar comandos e fornecem as informações necessárias para a execução daquele comando (Schneider, 1990).

A replicação de uma máquina de estado é geralmente implementada por meio da replicação de um *log*, que consiste de uma sequência de comandos que é executada pela máquina de estado. Cada réplica armazena um *log*. Cada réplica deve manter os mesmos comandos na mesma ordem, permitindo que a réplica execute a mesma sequência de comandos das outras réplicas que resulta no mesmo estado. A consistência dos *logs* nas diferentes réplicas é garantida por um módulo que implementa o consenso entre as réplicas, *i.e.*, garante o acordo entre as réplicas sobre a ordem em que os comandos devem ser executados. O módulo do consenso é responsável por receber os comandos dos clientes e replicá-los de forma consistente para as demais réplicas que participam do consenso.

A Figura 5.1 mostra um exemplo de máquina de estado replicada e como ocorre a replicação de seu *log* em um conjunto de três processos replicados. As aplicações cliente (*Clients*) solicitam a execução de comandos da máquina de estado para o módulo do consenso (*Consensus Module*). O módulo do consenso de uma réplica comunica com os módulos do consenso das outras réplicas para garantir que *log* que está sendo replicado esteja consistente entre os processos que participam do consenso.

No exemplo da Figura 5.1, uma aplicação cliente solicita a execução de um comando (passo 1) para a *Réplica 1*. Em seguida, o módulo do consenso replica o comando recebido para as demais réplicas utilizando comunicação *multicast* (passo 2). O objetivo é chegar a um acordo para aceitação do comando proposto pelo cliente. Após as réplicas chegarem a um acordo e decidirem pelo comando proposto (passo 3), a máquina de estado de cada réplica pode executar o comando que foi acordado. Como o módulo do consenso deve garantir a consistência do *log* replicado, todas as réplicas executam os mesmos comandos em ordem até o ponto em que o *log* das mesmas se encontra consistente. A execução de um comando geralmente resulta na atualização das variáveis de estado da máquina de estado (passo 4). A máquina de estado por sua vez, retorna o resultado para a aplicação cliente que solicitou a execução do respectivo comando (passo 5).

Este trabalho propõe o uso da replicação de máquina de estado dos xApps para implementar a tolerância a falhas no RIC. Entretanto, esta estratégia só pode ser aplicada se for possível manter a latência do RIC em no máximo 2ms, além da escalabilidade necessária para suportar dezenas de milhares de requisições por segundo (O-RAN Alliance, 2020c,a). O limite de 2ms para a latência do *loop* de controle do RIC é derivado da suposição de que uma requisição deve ser atendida em no máximo 10ms e a latência de comunicação entre o RIC e um determinado elemento da RAN pode chegar a 4ms em cada sentido da comunicação.

Argumentamos que a utilização de técnicas de *particionamento de estado com replicação parcial aproximada* e *re-roteamento com ciência de papel* permitem atender aos requisitos impostos pelo controlador RIC e a RAN. O particionamento de estado de um xApp, permite

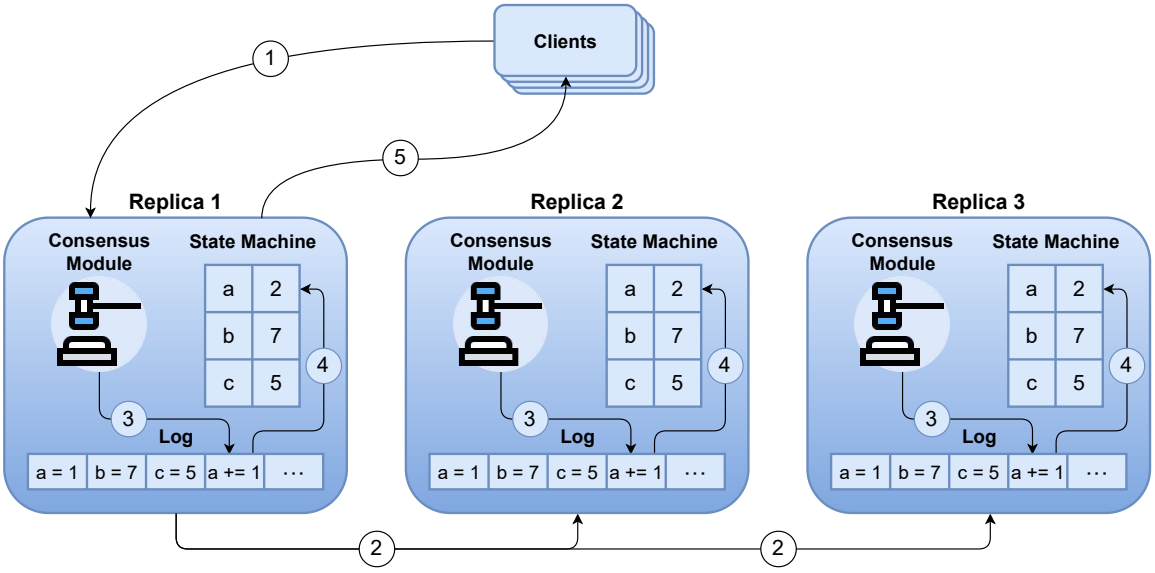


Figura 5.1: Exemplo de uma máquina de estado replicada.

que diferentes conjuntos de dados sejam mantidos por diferentes réplicas daquele mesmo xApp. Na replicação parcial, um conjunto de dados é mantido por uma réplica dita primária de um xApp. O mesmo xApp pode ter mais que uma réplica primária: diferentes réplicas primárias são responsáveis por dados distintos. Os dados de uma réplica primária são replicados para um grupo de réplicas ditas de *backup*. A quantidade de réplicas *backup* e as garantias de consistência podem ser configuradas, por exemplo a frequência de replicação. O re-roteamento com ciência de papel permite que uma mensagem seja rapidamente redirecionada para o *backup* adequado, sempre que for necessário acessar os dados correspondentes e a réplica primária estiver inacessível ou mesmo sobrecarregada. O *backup* então deve ser capaz de processar a mensagem com a ciência de que o estado pode não refletir exatamente ao estado da réplica primária.

5.2 XAPPS TOLERANTES A FALHAS

Esta seção descreve uma visão geral dos componentes do controlador RIC que são fundamentais para construir xApps tolerantes a falhas.

5.2.1 O RIC e os xApps

A arquitetura da O-RAN (O-RAN Alliance, 2020a) define que o RIC monitora e controla os elementos da RAN através da interface E2, que é utilizada pelos xApps para receber eventos específicos da RAN (*e.g.*, um novo UE tentando se conectar na rede) e para emitir mensagens de controle (*e.g.*, rejeitar um pedido de conexão ou mover um UE de uma célula para outra). Uma célula consiste em uma área geográfica limitada em que há cobertura de sinal da RAN. Cada célula é servida por um transceptor (*i.e.*, equipamento que é capaz de modular sinais de rádio) a partir de uma estação base (*Base Station* ou *Cell Site*). Cada transceptor gerencia um conjunto de antenas que apontam para diferentes direções (*i.e.*, setores) e utilizam diferentes faixas de frequência (Habibi et al., 2019). Um *Cell Site* compreende os equipamentos de rádio e de rede instalados em um determinado local, utilizados para transmitir e receber dados da RAN. No contexto deste trabalho, um elemento da RAN é qualquer componente da RAN que troca mensagens com os xApps através da interface E2.

Os xApps são executados no RIC e têm como principal objetivo fornecer funções de controle específicas e bem definidas que permitem customizar e otimizar o comportamento da RAN. As funções de controle fornecidas pelos xApps são consideradas funções adicionais e que, portanto, complementam as funcionalidades que já são fornecidas originalmente pelos elementos da RAN. Cada elemento da RAN exporta as suas funcionalidades para o RIC através da interface E2, a qual é utilizada pelos xApps para monitorar, suspender/parar, sobrescrever, ou mesmo controlar o comportamento padrão dos respectivos elementos da RAN (O-RAN Alliance, 2020a,b).

Cada instância do RIC pode gerenciar um número potencialmente grande de elementos da RAN em uma determinada região geográfica. Os elementos da RAN por sua vez, devem ser capazes de continuar provendo os seus serviços para a RAN mesmo em uma eventual falha completa do controlador RIC, resultando apenas na interrupção temporária das funções adicionais de customização e otimização fornecidas pelo RIC (O-RAN Alliance, 2020a,b). Exemplos de funcionalidades adicionais que o RIC pode fornecer através de xApps incluem prevenção de DDoS, transferência de UE entre células (*i.e.*, *handover*), previsão de congestionamento de células e otimização de QoS (*Quality of Service*) baseado na categoria de UEs (*e.g.*, equipamentos usados por policiais, dispositivos da IoT e veículos conectados). Cada xApp interage com os componentes da plataforma RIC e outros xApps usando duas interfaces principais:

- RMR (*RIC Message Router*): uma biblioteca utilizada pelos componentes do controlador RIC para trocar mensagens em uma mesma instância do RIC. O roteamento das mensagens é orientado por políticas baseadas em atributos da mensagem, tais como tipo da mensagem, endereços de origem e destino de elementos da RAN, componentes do RIC e xApps, e, identificador de assinatura que é utilizado no contexto da comunicação dentro do paradigma *Publish-Subscribe* e permite fazer "assinaturas" para receber dados específicos. A estratégia empregada pelo RMR permite que um remetente simplesmente preencha alguns atributos da mensagem e a envie sem necessariamente especificar o destino da mesma. A RFT aproveita esta estratégia para fornecer serviços de tolerância a falhas de forma transparente aos xApps; e,

- SDL (*Shared Data Layer*): fornece uma abstração de armazenamento *chave-valor* e permite aos xApps armazenar/recuperar informações fornecidas por outros componentes da plataforma RIC e também por outros xApps.

A Figura 5.2 ilustra os componentes do controlador RIC que são fundamentais para construir xApps tolerantes a falhas. O componente *E2 Term* implementa a interface para os elementos da RAN utilizando o protocolo de transporte SCTP (*Stream Control Transmission Protocol*). O *payload* das mensagens é especificado utilizando ASN.1 (*Abstract Syntax Notation 1*). O componente *Subscription Manager* implementa o padrão de troca de mensagens, que ocorre de acordo com o paradigma *Publish-Subscribe*. O *Subscription Manager* permite que xApps assinem tópicos de interesse que representam eventos específicos que ocorrem na RAN. O *Routing Manager* é responsável por atualizar dinamicamente as políticas de roteamento de mensagens nos componentes da plataforma RIC. A figura assume que xApps podem ser replicados; as n réplicas de um determinado xApp são referenciadas como $xApp_{1..n}$. É importante destacar que o RIC inclui vários componentes adicionais e que não são apresentados na Figura 5.2, estando incluídos apenas os componentes essenciais para descrever a RFT.

O RIC é executado em um cluster Kubernetes (Kubernetes, 2020) e seus componentes são implantados por meio de *Pods* que consistem de um ou mais contêineres. O cluster Kubernetes é geralmente executado em um cluster de servidores de processamento, e seus componentes são

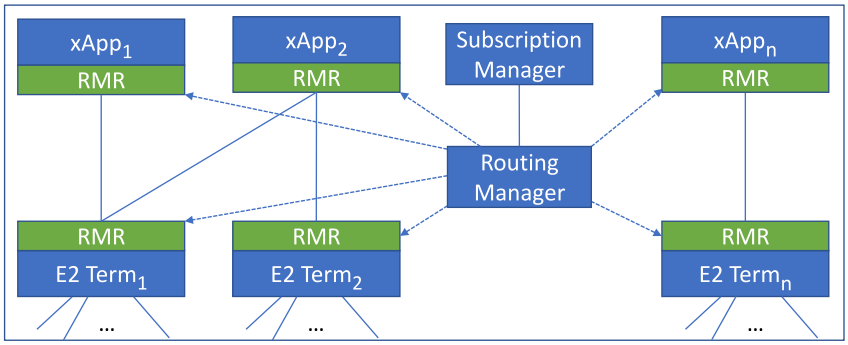


Figura 5.2: Componentes do controlador RIC.

replicados em diferentes servidores para evitar pontos únicos de falhas. A versão atual do RIC (*i.e.*, Cherry) não oferece suporte à replicação de xApps.

5.2.2 Informações de Estado dos xApps

As réplicas de um determinado xApp podem executar em máquinas distintas. Cada xApp é inicializado no RIC com um estado inicial. Os xApps podem ser classificados como *stateless* ou *stateful*. Um xApp *stateless* requer apenas as informações contidas em cada mensagem recebida para executar sua função. Deste modo, réplicas de xApps *stateless* podem processar qualquer mensagem em qualquer momento sem qualquer informação de estado anterior. Exemplos de funcionalidades implementadas em xApps que se encaixam na classe *stateless* incluem aqueles com funcionalidades tais como:

- traduzir uma mensagem codificada no formato ASN.1 recebida de um elemento da RAN em uma mensagem correspondente codificada em outro formato;
- decodificar e extrair informações do *payload* de uma mensagem (*e.g.*, intensidade de sinal, identificador do UE);
- utilizar a informação de uma mensagem decodificada para calcular uma nova métrica. Por exemplo, utilizar medições de intensidade de sinal de um UE em várias células vizinhas para estimar a localização daquele UE.

Desta forma, a replicação de xApps *stateless* é muito simples. Qualquer réplica pode processar qualquer requisição/mensagem. Assim, basta utilizar um algoritmo de balanceamento de carga para distribuir o tráfego entre as várias réplicas. O requisito básico, neste caso, é que uma réplica falha seja detectada o mais breve possível para que as decisões sejam tomadas levando em conta as réplicas corretas.

Por outro lado, um xApp *stateful* depende de informações de estado para processar cada nova mensagem. O estado de uma réplica individual consiste de um ou mais *contextos*. Um *contexto* inclui informações sobre um elemento específico da RAN. As informações específicas de cada contexto estão diretamente relacionadas com a lógica de cada xApp, e são determinadas pelo desenvolvedor do xApp. Exemplos de contexto incluem: o identificador de um UE, de um grupo de UEs, uma célula específica da RAN, uma estação base (*i.e.*, *Cell Site*), uma categoria de UE (*e.g.*, dispositivos IoT, telefones celulares), entre outros. Por sua vez, o estado de um xApp consiste do conjunto de contextos mantido por todas as réplicas do xApp. Cada réplica do xApp pode manter informações de contexto referentes a vários elementos da RAN. Réplicas distintas podem manter contextos distintos.

Cada contexto tem um identificador único. As informações mantidas em um contexto são estruturadas por meio de pares *chave-valor*. Como exemplo, considere um contexto de um determinado xApp referente a uma célula. Neste caso, o identificador do contexto é o próprio identificador daquela célula na RAN. O contexto pode ter informações diversas sobre a célula, tais como o nível de sinal e a utilização, que são armazenados como pares *chave-valor*.

Diversos xApps requerem informações de contexto para processar uma mensagem e, portanto, se encaixam na classe *stateful*. Exemplos de funcionalidades implementadas por xApps *stateful* e as respectivas informações de contexto incluem:

- medir a duração da conexão de um UE: é necessário armazenar tanto o instante de estabelecimento como de encerramento da conexão daquele UE;
- reportar o número de UEs conectadas em uma determinada célula ou em uma região coberta por várias células: é necessário o recebimento das mensagens de estabelecimento e encerramento de conexão de UEs para contabilizar o número presente em um determinado instante de tempo;
- prever a utilização de uma determinada célula: a partir da utilização é possível por exemplo evitar sobrecarga, movendo determinados UEs para células vizinhas (*i.e.*, *handover*). Informações de contexto da célula de destino e da células vizinhas são necessárias.

Os contextos de um xApp podem ser replicados de diferentes maneiras. Por exemplo, todos os contextos poderiam ser replicados em todas as réplicas do xApp. Entretanto, dependendo da quantidade de informação dos contextos e da frequência com que itens diferentes de contextos diferentes são atualizados, esta solução pode ter custo muito alto em termos de desempenho. Assim, a estratégia proposta nesta Tese é *particionar* o estado de um xApp em contextos e *replicá-los parcialmente* entre as instâncias do xApp. O particionamento permite duas réplicas distintas de um xApp manter conjuntos diferentes de contextos do xApp. Particionar o estado dessa forma permite reduzir a quantidade de mensagens que uma instância de um xApp precisa processar. Diminui também o tempo que a instância precisa para atualizar seu estado a cada nova requisição recebida.

Além do particionamento, que limita a quantidade de contextos que uma réplica de um xApp mantém, a *replicação parcial* de contextos permite que um xApp tenha um conjunto configurável de réplicas. Antes, uma definição: uma réplica primária é responsável por manter e atualizar um contexto enquanto uma réplica *backup* é responsável apenas por manter uma cópia do contexto. Neste trabalho, os termos *primário* e *backup* também são usados como sinônimos para réplica primária e réplica *backup*, respectivamente. Na replicação parcial, cada réplica do xApp pode ser primária para determinados contextos e pode ter um número configurável de *backups* daquele contexto entre as demais réplicas do xApp.

A Figura 5.3 apresenta um exemplo de replicação parcial de contextos em que um xApp possui três réplicas ($xApp_1$, $xApp_2$ e $xApp_3$) e mantém um *backup* para cada um dos quatro contextos. Cada uma das réplicas pode ser primária para certos contextos e *backup* para outros. No exemplo da figura, a réplica $xApp_1$ é primária para os contextos *c1* e *c2* enquanto é *backup* do contexto *c4*. Por outro lado, a réplica $xApp_2$ é primária do contexto *c3* e também é *backup* dos contextos *c1* e *c2*. Assim, um determinado contexto é replicado apenas para um conjunto parcial de todas as réplicas de um determinado xApp, *i.e.*, somente uma réplica *backup* de duas réplicas disponíveis neste exemplo.

Embora a replicação garanta que o serviço provido por um xApp esteja disponível mesmo após a ocorrência de falhas, a replicação por si só não assegura que o RIC é capaz de processar e

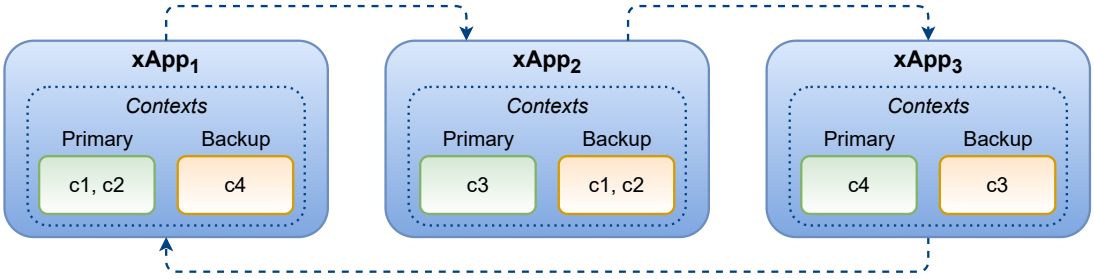


Figura 5.3: Exemplo de replicação parcial de contextos.

responder uma mensagem com uma requisição referente a um contexto que chega logo após sua réplica primária ter falhado. O *re-roteamento com ciência de papel* tem essa funcionalidade, ou seja, possibilita que a mensagem seja direcionada rapidamente para uma réplica *backup*. Especificamente, caso a réplica primária de um determinado contexto não esteja disponível para processar a mensagem de um elemento da RAN, a mensagem é imediatamente encaminhada para uma das réplicas *backup* que possui uma cópia aproximada daquele contexto. O re-roteamento ciente de papel é usado para re-encaminhar mensagens após falhas. Por exemplo, seja a réplica X de um determinado xApp primária para um determinado contexto, é possível configurar que Y e Z são *backups* daquele contexto. Caso a réplica X falhe, as mensagens são re-encaminhadas para outra réplica, por exemplo Y, que processa a requisição na mensagem adequadamente. A réplica *backup* que recebe a mensagem da RAN estará ciente de sua função como *backup* para aquele contexto e pode usar esta "ciência" para processar a mensagem considerando que suas informações de contexto podem não estar totalmente atualizadas.

Mesmo a replicação parcial também pode ser cara em termos de latência caso seja realizada de maneira síncrona (*i.e.*, qualquer atualização de contexto é imediatamente processada pela réplica primária e pelas réplicas *backup*). Visando reduzir a latência dos xApps, é proposta a replicação assíncrona de contextos, em que inicialmente, apenas a réplica primária atualiza o contexto, o que é feito posteriormente pelas réplicas *backup*. A replicação assíncrona permite que a réplica primária possa modificar as informações dos contextos e responder as requisições da RAN sem ter que aguardar confirmações das réplicas *backup*. Na estratégia de replicação proposta, após a falha de uma réplica primária para um determinado contexto, uma das réplicas *backup* daquele contexto é eleita como sua nova primária.

5.3 RFT: *RIC FAULT TOLERANCE*

O objetivo da RFT (*RIC Fault Tolerance*) é definir um conjunto de técnicas para a implementação de xApps tolerantes a falhas, capazes de sustentar baixa latência e manter a escalabilidade necessária para suportar dezenas de milhares de requisições por segundo. Como mencionado na seção anterior, a solução consiste de técnicas de particionamento de estado com replicação parcial e re-roteamento com ciência de papel. A RFT é implementada e disponibilizada por meio de uma biblioteca (Huff et al., 2020) que pode ser vinculada à implementação dos xApps da mesma forma que outras bibliotecas da plataforma RIC, tais como RMR, SDL e bibliotecas de *logging*.

A RFT é baseada em uma estratégia de *group membership* (Cachin et al., 2011). Um sistema de *group membership* permite que os membros do grupo tenham uma visão consistente sobre a composição do grupo. Desta forma, o consenso é usado para que haja um acordo sobre a composição, *i.e.*, o conjunto de réplicas corretas em um determinado instante de tempo. A estratégia utilizada pela RFT possibilita a gerência da composição do grupo de réplicas de xApps

e o roteamento de mensagens que se adapta ao conjunto de membros corretos de um determinado grupo.

xApps tolerantes a falhas podem ser construídos de forma transparente por meio da RFT, *i.e.*, o fato de um xApp ser replicado fica transparente para o restante do sistema. Vale ressaltar que simplesmente depender do Kubernetes para adicionar tolerância a falhas aos xApps não é uma alternativa viável. Embora o Kubernetes faça monitoramento de *Pods* (*i.e.*, um grupo de um ou mais contêineres executando em um *cluster*), este monitoramento não é suficiente para implementar toda a lógica de replicação de xApps. Por exemplo, após a detecção da mudança de estado de um membro do grupo, pode ser necessário modificar os papéis (*i.e.*, primário ou *backup*) de determinadas réplicas do xApp. Também são necessárias atualizações nas políticas de roteamento para encaminhar as mensagens da RAN corretamente às novas réplicas primária e *backup*.

A RFT implementa a gerência de grupos de réplicas de xApps utilizando o algoritmo de consenso Raft (Ongaro e Ousterhout, 2014; Ongaro, 2014). O algoritmo Raft permite manter a sequência do *log* da máquina de estado consistente em cada uma das réplicas. Um dos componentes principais do Raft é referente à eleição de um líder entre os processos que executam o consenso. Assim, a consistência do *log* é garantida através do processo líder que gerencia a replicação de comandos do *log* para os demais processos. O fluxo de mensagens de replicação do Raft é sempre executado do processo líder para os demais processos que participam do consenso. O líder também aceita comandos de processos cliente que são adicionados em sequência no *log* e replicados para os demais processos do consenso. Os comandos somente são aplicados (*i.e.*, executados) em cada máquina de estado após o líder confirmar que o comando foi devidamente replicado na maioria dos processos corretos que participam do consenso.

Cada comando que é executado pela máquina de estado é armazenado no *log* de cada uma das réplicas. Para evitar que o *log* cresça indefinidamente e permitir a manipulação de seus dados de forma eficaz, o Raft define o conceito de *snapshot*. Um *snapshot* consiste do conjunto atualizado de valores de todas as variáveis que compõem o estado. Um *snapshot* reúne os dados efetivamente necessários para comunicação entre réplicas, para a qual é serializado. A Figura 5.4 apresenta um exemplo de uma máquina de estado que mantém três variáveis (*i.e.*, *a*, *b* e *c*). Um *snapshot*, neste exemplo, consiste na serialização dos valores *2*, *7* e *5* respectivamente para as variáveis de estado *a*, *b* e *c*. A estratégia de serialização de *snapshot* pode diferir de acordo com a implementação do algoritmo de consenso (Ongaro e Ousterhout, 2014). Ao ser recebido no destino, o *snapshot* é desserializado, processo que neste trabalho é chamado de "instalação de um *snapshot*".

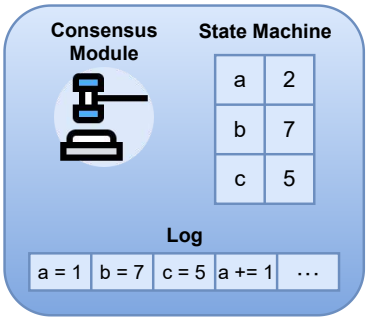


Figura 5.4: Exemplo de uma máquina de estado com três variáveis de estado.

5.4 GERENCIAMENTO DE GRUPOS DE RÉPLICAS

O gerenciamento do grupo de réplicas de um xApp implementado na RFT permite que as réplicas corretas do xApp tenham uma visão consistente de todos os membros corretos que participam do mesmo grupo. Um grupo de réplicas de um xApp consiste do conjunto de réplicas de um determinado xApp que está em execução em uma única instância do RIC. O algoritmo de consenso Raft implementado na RFT permite manter uma lista de membros corretos de um grupo consistente em todas as réplicas que pertencem ao mesmo grupo do xApp. Ao invés de utilizar um sistema externo que oferecesse o serviço de consenso (por exemplo o Zookeeper (Apache, 2020b)) optou-se por implementar o Raft na própria RFT com o objetivo de simplificar o desenvolvimento e implantação de xApps tolerantes a falhas usando o mínimo de dependências externas.

Uma das réplicas é eleita líder do grupo de réplicas de um determinado xApp. A eleição de líder é executada pelo algoritmo de consenso Raft. A réplica que foi eleita líder do grupo tem a função de monitorar as demais réplicas que executam o consenso para a detecção de falhas. O monitoramento das réplicas é realizado por meio de mensagens de *heartbeat* enviadas pelo líder periodicamente para as réplicas que são membros do mesmo grupo. Estas réplicas são chamadas de *Raft Followers* e respondem as mensagens de *heartbeat* para o líder. Inicialmente, as réplicas que são *Raft Followers* iniciam uma nova eleição de líder (executando o algoritmo de eleição de líder do Raft, tornando-se *Raft Candidates*). O mesmo procedimento é também executado toda vez que haja suspeita de que o líder esteja falho. O líder por sua vez, suspeita que uma determinada réplica do grupo esteja falha após esta réplica não responder um determinado número consecutivo de mensagens *heartbeat*. O líder ainda é responsável por adicionar e remover as réplicas do xApp que pertencem ao mesmo grupo do líder.

A Figura 5.5 mostra como uma nova réplica é adicionada a um grupo de réplicas (*membership*) de um xApp e como as respectivas políticas de roteamento são atualizadas. A figura mostra a adição de uma nova réplica ($xApp_3$) a um grupo de réplicas que já contém a réplica $xApp_1$ e $xApp_2$. A figura mostra também o componente do RIC que faz a gerência de roteamento (*Routing Manager*) e o componente $E2\ Term$ que corresponde à interface de comunicação entre os xApps e os elementos da RAN. Desta forma, o contexto mantido pelo xApp é referente a um elemento da RAN que usa o componente $E2\ Term$ para enviar mensagens para as réplicas do xApp. Considera-se que o líder ($xApp_1$) foi eleito previamente entre os membros do grupo. As réplicas do xApp são monitoradas por meio de mensagens de *heartbeat*, que não são mostradas na figura.

Inicialmente, o $xApp_3$ envia mensagens para os membros atuais do grupo solicitando a sua adesão (mensagens *1* e *2*). Ao receber esta mensagem, o líder do grupo envia a mensagem *Install Snapshot Request* (mensagem *3*) para o $xApp_3$ com as informações de estado atual do grupo (*i.e.*, informações de que atualmente o $xApp_1$ e $xApp_2$ estão no grupo). O *snapshot*, neste caso, consiste do estado atual do grupo serializado e é enviado pelo líder sempre que um novo membro solicita a entrada naquele grupo. Depois que o $xApp_3$ finaliza a desserialização do *snapshot*, o $xApp_3$ envia uma mensagem ao líder confirmando que o *snapshot* foi instalado (mensagem *4*). Esta mensagem também indica que o líder pode iniciar o monitoramento do $xApp_3$.

Em seguida, a solicitação de adesão ao grupo realizada pelo $xApp_3$ é replicada pelo líder para todos os outros membros do grupo (neste caso, apenas para o $xApp_2$), conforme mostrado na mensagem *5*. Após o líder receber a confirmação da replicação da maioria dos membros do grupo (mensagem *6*), o líder efetiva (*i.e.*, *commit*) as informações locais de estado do agrupamento (mensagem *7*). Na sequência, o líder replica as atualizações de estado do grupo

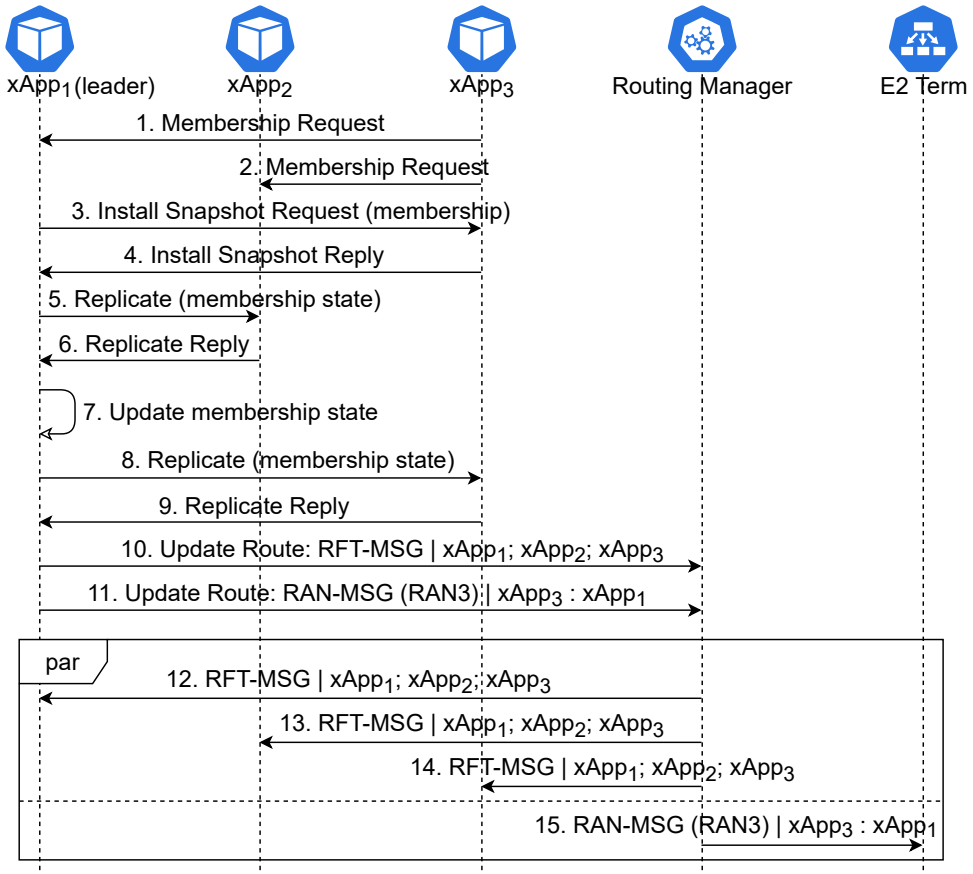


Figura 5.5: Uma nova réplica é adicionada ao grupo de um xApp e as políticas de roteamento são atualizadas.

para o $xApp_3$, que por sua vez toma o conhecimento de que foi adicionado ao grupo de réplicas do xApp. As mensagens *8* e *9* representam a replicação da atualização de estado do grupo para o $xApp_3$.

Após o $xApp_3$ ter sido incluído no grupo, o líder atualiza as políticas de roteamento do RMR para permitir que as mensagens de elementos da RAN e mensagens de gerência do grupo também sejam encaminhadas para o $xApp_3$. A RFT emprega dois tipos de mensagens. As mensagens do tipo RFT-MSG são usadas pelo subsistema de agrupamento de xApps da RFT, enquanto que as mensagens do tipo RAN-MSG são usadas pelos elementos da RAN para enviar mensagens aos xApps. Toda comunicação entre as réplicas de um determinado grupo é feita através de *multicast*. Com a inclusão da nova réplica é necessário atualizar os destinatários de cada mensagem *multicast*, o que é solicitado através da mensagem *Update Route* que é enviada para o componente *Routing Manager* (mensagem *10*). Esta atualização informa ao RMR que todas as mensagens do tipo RFT-MSG devem ser entregues utilizando *multicast* (*i.e.*, indicado pelos pontos-e-virgulas) para as réplicas da RFT que executam no $xApp_1$, $xApp_2$, e $xApp_3$.

O líder ainda envia outra mensagem *Update Route* para o *Routing Manager* especificando que todas as mensagens do tipo RAN-MSG originadas pelo elemento da RAN identificado por *RAN3* sejam enviadas para a réplica primária $xApp_3$ (*considera-se que os contextos do elemento RAN3 tenham sido alocados para o* $xApp_3$ *na execução da mensagem 7*). Caso a réplica $xApp_3$ esteja indisponível, a mensagem é re-roteada para a réplica *backup* $xApp_1$ (mensagem *11*). Ao receber as mensagens *10* e *11*, o *Routing Manager* atualiza as políticas de roteamento nos respectivos membros do grupo do xApp e nos componentes do RIC. Depois que a nova política de *multicast* foi instalada (mensagens *12-14*), cópias de cada mensagem do tipo RFT-MSG são

enviadas para o $xApp_1$, $xApp_2$, e $xApp_3$. Após a entrega da mensagem *15* ao componente $E2\ Term$, todas as mensagens do tipo RAN-MSG originadas do elemento *RAN3* são entregues à réplica primária $xApp_3$, contudo, podem ser entregues à réplica $xApp_1$ caso $xApp_3$ estiver inacessível.

Depois de receber uma mensagem de replicação com a atualização do grupo de réplicas (*i.e.*, nova configuração), cada réplica do xApp seleciona as réplicas *backup* que devem manter cópia de seus contextos primários. As réplicas *backup* são selecionadas com base na sequência atualizada da lista de membros do grupo do xApp. Cada réplica primária do xApp seleciona as suas respectivas réplicas *backup* da seguinte maneira. Seja *m* o número de réplicas *backup* configuradas para um determinado xApp, as próximas *m* réplicas disponíveis na lista de membros do grupo são selecionadas como réplicas *backup*.

A Figura 5.6 mostra como uma réplica falha é removida do grupo de réplicas de um xApp e as respectivas políticas de roteamento são atualizadas. Assume-se nesta figura que o contexto mantido pelas réplicas é referente a um elemento da RAN. Também assume-se que a réplica da RFT que executa no $xApp_1$ é líder do grupo e foi eleita anteriormente. O líder do grupo envia mensagens *heartbeat* para os outros membros do grupo conforme ilustrado pelas mensagens *1* e *3*. As réplicas da RFT que executam no $xApp_2$ e $xApp_3$ são *Raft Followers* e respondem mensagens de *heartbeat* para o líder (apenas a mensagem *2* neste exemplo).

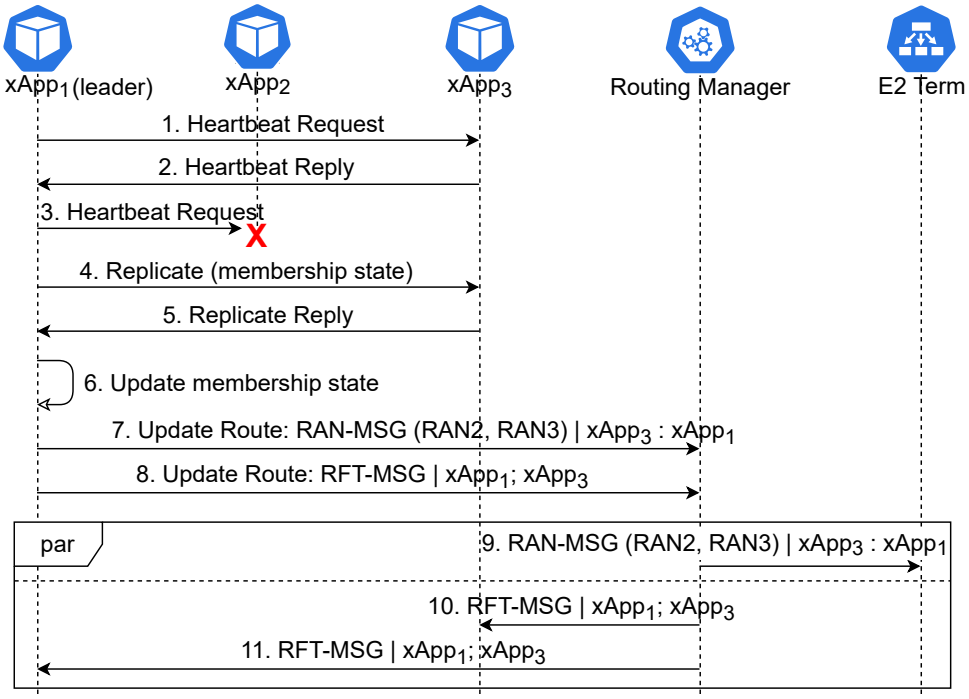


Figura 5.6: Uma réplica é removida do grupo de um xApp e as políticas de roteamento são atualizadas.

O líder suspeita que uma réplica do xApp esteja falha caso não receba nenhuma mensagem de *heartbeat* da respectiva instância da RFT dentro de um tempo limite configurável (mensagem *3*). Ao suspeitar que uma réplica do xApp esteja falha, o líder replica as alterações do grupo (mensagem *4*) para as demais réplicas do grupo que não falharam ($xApp_3$ neste exemplo). Após o líder receber a confirmação da mensagem de replicação da maioria do membros do grupo (mensagem *5*), o líder efetiva (*i.e.*, *commit*) a remoção da réplica falha do grupo de réplicas do xApp (mensagem *6*). Assume-se na figura que o líder realoca os elementos da RAN e os respectivos contextos do $xApp_2$ para o $xApp_3$.

Depois que o $xApp_2$ foi removido do grupo, o líder solicita a atualização das políticas de roteamento para o *Routing Manager* de forma que as mensagens da RAN que estavam sendo endereçadas para o $xApp_2$ sejam endereçadas para o $xApp_3$ (mensagem *7*). O líder também solicita para o *Routing Manager* atualizar a política de roteamento *multicast* removendo a réplica $xApp_2$ do grupo de réplicas do xApp (mensagem *8*). O *Routing Manager* então atualiza as políticas de roteamento do componente $E2\ Term$ através da mensagem *9*. Esta mensagem determina que todas as requisições do tipo RAN-MSG originadas pelos elementos *RAN2* e *RAN3* sejam enviadas para a réplica $xApp_3$ (primária). Caso a réplica $xApp_3$ esteja indisponível, as requisições são enviadas para a réplica $xApp_1$ (*backup*). O *Routing Manager* também atualiza o roteamento *multicast* entre os membros do grupo que estão sem falha enviando as mensagens *10* e *11*, incluindo o líder. As políticas de roteamento das mensagens *10* e *11* indicam que todas as mensagens do tipo RFT-MSG devem ser enviadas para as réplicas $xApp_1$ e $xApp_3$. É importante evidenciar que as mensagens *9-11* são assíncronas, no sentido de que elas podem ser entregues em ordens diferentes.

Na versão atual da RFT, dado um grupo de réplicas, a réplica que recebe o primeiro comando de replicação de um determinado contexto é configurada primária daquele contexto, as demais são *backup*, considerando o número de réplicas *backup* configuradas para aquele grupo. No futuro é possível adotar outras estratégias para a escolha, levando em conta questões de desempenho, uso de recursos, latência e outras.

5.5 REPLICAÇÃO PARCIAL DE ESTADOS

A RFT explora uma estratégia de replicação parcial dos xApps. A replicação parcial permite a cada réplica de um grupo de xApps replicar seus contextos primários a um conjunto de réplicas *backup* do mesmo grupo. O número de réplicas *backup* e a frequência de replicação são configuráveis. Após ocorrer um evento na RAN (*e.g.*, um UE se conecta a uma célula da rede), uma mensagem de notificação pode ser enviada para a réplica primária do xApp que realizou a assinatura do respectivo evento. Caso o primário estiver indisponível, uma das réplicas *backup* recebe a mensagem de notificação. Além disso, alguns eventos que ocorrem na RAN podem ser processados diretamente pelos próprios elementos da RAN, enquanto que outros eventos são enviados para o RIC. Assim, algumas informações de estado da RAN podem ser mantidas internamente em um elemento da RAN, outras nos xApps ou ainda em ambos (*i.e.*, elemento da RAN e xApps) (O-RAN Alliance, 2020b).

Diante disso, os xApps devem ser capazes de executar as suas funções de controle utilizando informações de estado locais e que podem ser aproximadas se comparadas com o estado atual da RAN (O-RAN Alliance, 2020a,b). Considere por exemplo, que um xApp esteja monitorando eventos de conexão de UEs de um conjunto de células vizinhas com o objetivo de balancear a quantidade de conexões de UEs entre as mesmas. Especificamente, um elemento da RAN pode receber várias requisições simultâneas de conexão de diferentes UEs em diferentes células em um curto período de tempo que, consequentemente, alteram as informações de estado locais no elemento da RAN. Assim, o estado de uma réplica do xApp – mesmo primária – pode não refletir o estado atual do elemento da RAN (*e.g.*, quantidade de UEs conectados em cada célula) após a mensagem ter sido processada pela réplica do xApp, pois novas conexões podem ter ocorrido na RAN mas que ainda não tenham sido notificadas a tempo ao xApp.

A RFT aproveita a característica que os xApps apresentam de processar as mensagens da RAN com estado aproximado para implementar a replicação parcial de estados dos xApps. Na RFT, as réplicas primárias atualizam e replicam os contextos primários para as réplicas *backup*. As réplicas *backup* por sua vez, não modificam as informações dos contextos que são cópia de

um primário, ou seja, as cópias dos contextos são somente leitura. Assim, a consistência entre as réplicas é garantida, pois o fluxo de mensagens de replicação é executado da réplica primária para as réplicas *backup*.

Um ponto importante a mencionar é que a replicação parcial dos xApps não é baseada no algoritmo Raft. O Raft é empregado para a replicação das informações dos membros do grupo de um determinado xApp. Concretamente, na replicação parcial de estados, são replicados os comandos executados pela máquina de estado que corresponde ao xApp. xApps diferentes são máquinas de estado diferentes. A unidade básica de dados que é replicada é chamada de *log entry* e consiste nos seguintes atributos:

- *cmd*: consiste no comando da máquina de estado do xApp e que deve ser executado na mesma ordem pela réplica primária e *backups* atualizando um determinado contexto;
- *context*: representa o contexto ao qual a mensagem da RAN está associada. Este atributo consiste no identificador único que representa um determinado elemento ou grupo de elementos da RAN. Exemplos de contexto incluem o identificador de um UE, de uma célula, de uma categoria de UEs, entre outros;
- *key*: as informações mantidas em um contexto são estruturadas por pares de *chave-valor*. O atributo *key* representa a chave que é utilizada para identificar uma informação específica de um contexto;
- *value*: consiste na informação necessária para executar o comando da máquina de estado definido em *cmd*. O atributo *value* representa o *valor* referente a um par *chave-valor*;
- *vlen*: representa o tamanho em bytes do atributo *value*. O tamanho do *value* é necessário uma vez que a RFT é agnóstica em termos de dados replicados pelo xApp. O atributo *value* consiste apenas de uma sequência de bytes para a RFT; e,
- *sequence*: consiste de um número de sequência exclusivo que identifica cada *log entry* replicado. Este atributo possibilita que as réplicas primária e *backup* possam controlar a sequência de *log entries* que foram replicadas.

Além disso, cada mensagem de replicação pode transportar vários *log entries* permitindo reduzir a sobrecarga dos cabeçalhos das mensagens de replicação e também o número de mensagens de confirmação. Uma mensagem de replicação enviada de uma réplica primária para um *backup* é chamada de *Replication Request*. A mensagem de confirmação da replicação enviada de uma réplica *backup* para o primário é chamada de *Replication Reply*.

Um cenário de replicação parcial de estados é ilustrado na Figura 5.7. As mensagens são trocadas entre três réplicas de um determinado xApp (cada réplica do xApp executa uma instância da RFT). Nesse cenário de replicação, pode-se observar que embora a instância da RFT em $xApp_1$ replique contextos para $xApp_2$, ela também é réplica *backup* do $xApp_3$. Como o $xApp_2$ possui uma cópia dos contextos do $xApp_1$, o $xApp_2$ também pode processar mensagens originalmente endereçadas ao $xApp_1$, mas com a ciência de que é *backup*.

Ao receber uma mensagem *Replication Request* do primário ($xApp_1$), o *backup* ($xApp_2$) compara se o número de sequência do primeiro *log entry* recebido na mensagem atual é exatamente igual ao próximo número de sequência esperado. O número de sequência esperado corresponde ao maior número de sequência de *log entries* recebido até o momento, incrementado em uma unidade. Essa verificação permite ao *backup* determinar se deve, ou não, aplicar a sequência de comandos que foram recebidos nos *log entries* na máquina de estado do xApp, atualizando os contextos correspondentes.

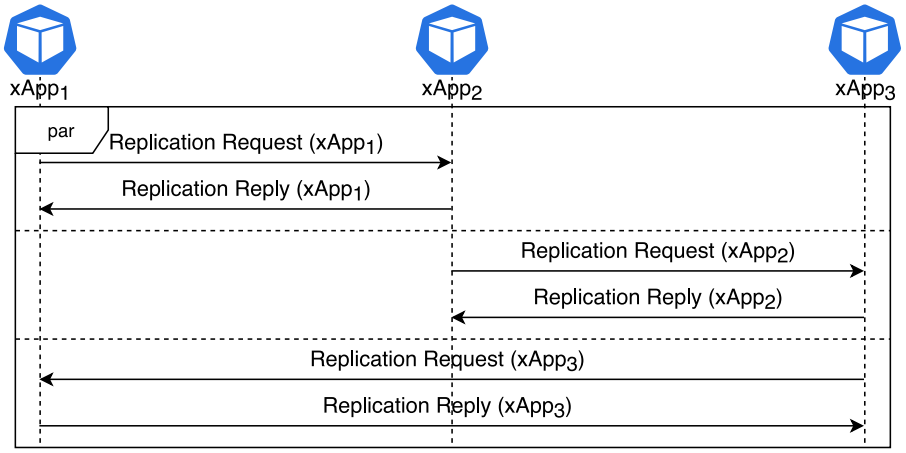


Figura 5.7: Replicação parcial de estados: $xApp_2$ é réplica *backup* do $xApp_1$ e $xApp_3$ é réplica *backup* do $xApp_2$.

Após executar os comandos recebidos, o *backup* envia uma mensagem *Replication Reply* para o primário confirmando o maior número de sequência dos *log entries* que foram replicados. Esta informação é então usada pelo primário para determinar o próximo *log entry* que será replicado para o *backup*. Caso o *backup* receber uma mensagem *Replication Request* com um número de sequência inesperado, os *log entries* recebidos são descartados e uma mensagem *Replication Reply* é enviada para o primário informando o maior número de sequência de *log entries* replicados até o momento.

Na medida em que novos *log entries* são replicados pela réplica primária, estes são adicionados ao *log* da máquina de estado do xApp que pode crescer indefinidamente. Para evitar que o tamanho deste *log* aumente de forma excessiva, a RTF cria *snapshots* periodicamente em cada réplica do xApp. *Snapshots* são usados pela réplica primária para transferir informações de contexto para um *backup*, descritos a seguir.

Um *snapshot* de uma determinada réplica do xApp para um determinado contexto consiste dos mais recentes *log entries* do *log* correspondente. Um *snapshot* tem tamanho máximo pré-definido, e é serializado antes de ser transmitido. O tamanho máximo do *snapshot* é configurável e depende inclusive do tamanho dos *log entries*. O tamanho de cada *log entry* varia conforme a quantidade de bytes serializados, principalmente do atributo *value*, que cada comando da máquina de estado carrega. Um novo *snapshot* é criado sempre que a soma dos tamanhos dos *log entries* produzidos até o momento atingir o limite pré-definido. Dessa forma, um *snapshot* estará disponível para transmissão por uma réplica primária para uma réplica *backup*.

Idealmente, um *snapshot* equivale a um grande número de atualizações de estado (*i.e.*, *log entries*). A estratégia de enviar *snapshots* economiza largura de banda de rede e ciclos de CPU tanto no remetente quanto no receptor, se comparada à alternativa de fazer o mesmo com *log entries* individuais. Pois basta serializar/desserializar um único *snapshot* ao invés de uma grande quantidade de *log entries*.

Os *snapshots* anteriores podem ser descartados após a criação de um novo *snapshot*. Os *snapshots* também permitem manter um *log* enxuto, descartando os *log entries* antigos e desnecessários. O descarte dos *log entries* é chamado neste trabalho de *log cleaning*. Depois que um novo *snapshot* é criado, todas as atualizações anteriores e, portanto, todos os *log entries* até aquele ponto não são mais necessários. A API disponibilizada pela RFT para a implementação de xApps tolerantes a falhas é apresentada no Apêndice B.

5.6 ARQUITETURA DA RFT

Nesta seção é apresentada a arquitetura da RFT, organizando todos os componentes apresentados nas seções anteriores. A Figura 5.8 apresenta a arquitetura da RFT e seus componentes, bem como a relação em alto nível entre a RFT, o RMR e os xApps (*xApp Logic*). Observa-se que ambos o xApp e a RFT compartilham a mesma instância do RMR para enviar e receber mensagens na rede.

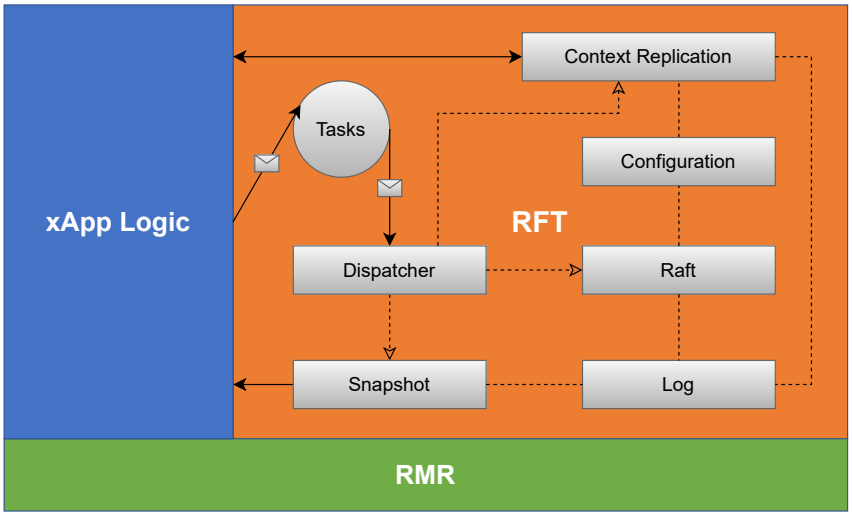


Figura 5.8: Arquitetura da RFT.

O componente *Tasks* consiste de uma fila de tarefas que a RFT deve executar em ordem. As tarefas são representadas nesta fila por meio de mensagens recebidas pela rede e que foram adicionadas por uma réplica do xApp. As tarefas desencadeiam diferentes operações na RFT, tais como adicionar um novo membro ao grupo de xApps ou processar uma mensagem de replicação. A fila de tarefas é assíncrona, no sentido de que as mensagens que estão nesta fila não bloqueiam o processamento de outras mensagens na réplica do xApp. Novas mensagens (*i.e.*, tarefas) da RFT podem ser adicionadas nessa fila mesmo que as mensagens anteriores não tenham sido completamente processadas pela RFT.

O componente *Dispatcher* distribui as tarefas entre os diferentes componentes da RFT com base no tipo da mensagem recebida. O tipo da mensagem identifica se a mesma é referente a replicação de contexto, uma solicitação de adição de membro no grupo, uma mensagem de monitoramento da RFT, entre outras. Outra funcionalidade do *Dispatcher* é desserializar o *payload* das mensagens recebidas antes de serem encaminhadas para os devidos componentes.

As mensagens referentes ao monitoramento de membros do grupo de xApps são encaminhadas ao componente chamado *Raft*, que implementa o algoritmo de consenso Raft (Ongaro, 2014). O componente *Raft* permite manter uma visão consistente dos membros corretos de um grupo de réplicas de um determinado xApp. O *Raft* replica os comandos da máquina de estado que modificam a composição do grupo de réplicas de um xApp e também realiza a eleição de líder entre as réplicas do grupo. Além disso, o componente *Raft* implementado na RFT permite adicionar e remover membros do grupo de xApps automaticamente, enquanto que a versão original do algoritmo Raft requer intervenção manual (Ongaro, 2014). A adição e remoção de membros do grupo foi descrita na Subseção 5.4.

O componente *Configuration* é essencialmente uma extensão do componente *Raft*. O componente *Configuration* mantém as informações sobre a composição de um grupo de réplicas de um xApp e é responsável por selecionar as réplicas *backup* de uma determinada réplica

primária do xApp. Uma lista com as réplicas *backup* é disponibilizada para o componente *Context Replication*.

O componente *Context Replication* basicamente possui duas responsabilidades: replicar as atualizações de contextos das réplicas primárias para as respectivas réplicas *backup*; e, aplicar as atualizações de contextos nas réplicas *backup* assim que foram recebidas da réplica primária. O componente *Context Replication* atualiza as informações de contexto nas réplicas *backup* através da função de *callback apply*. A *callback apply* é implementada no xApp (*xApp Logic*) e é descrita no Apêndice B. A RFT é agnóstica com relação às informações mantidas nos contextos e processadas pelos xApps. Cada xApp possui contextos diferentes com informações distintas e, que são processadas de formas específicas.

O componente *log* mantém em ordem os comandos executados e replicados pela máquina de estados, incluindo aqueles para gerenciar o grupo de réplicas de um xApp. Os comandos das máquinas de estados são mantidos em duas filas independentes. Uma fila mantém os comandos da máquina de estado dos contextos dos xApps que é replicada pelo componente *Context Replication*. Outra fila mantém os comandos da máquina de estados do grupo de réplicas do xApp que é replicada pelo componente *Raft*. Como descrito acima, os componentes *Context Replication* e *Raft* implementam estratégias diferentes de replicação. O componente *Context Replication* replica os comandos da máquina de estados dos contextos dos xApps empregando replicação parcial assíncrona e não aguarda confirmações das réplicas *backup* para responder as requisições de clientes. O componente *Raft* executa o algoritmo de consenso Raft que aguarda as confirmações da maioria das réplicas do grupo de um xApp, para então responder as requisições de clientes. Além disso, o componente *log* é responsável pela serialização do *payload* das mensagens de replicação para ambas as estratégias de replicação. O componente *Snapshot* por sua vez, é responsável coordenar o processo de serialização/desserialização dos *snapshots*. Em seguida, é apresentada uma avaliação empírica da RFT.

5.7 AVALIAÇÃO EMPÍRICA

Uma avaliação empírica foi realizada para avaliar o desempenho da RFT usando um xApp seletor de células (*Cell Selector xApp*) implementado em linguagem C para resolver um problema prático de seleção de células da RAN. O objetivo deste xApp é selecionar a melhor célula de destino em operações de *handover* solicitadas por diferentes UEs e melhorar o desempenho geral da rede. Especificamente, assume-se que na operação de *handover* a RAN irá gerar solicitações de transferência de UE sempre que um UE observar uma célula vizinha com intensidade maior de sinal do que a sua célula atual. No entanto, o fato de uma célula ter a maior intensidade de sinal não a torna necessariamente a melhor escolha para transferir o UE para a mesma. Por exemplo, a célula pode estar congestionada devido a um grande número de UEs conectados em um determinado momento. O *Cell Selector xApp* considera também a utilização de recursos das células que pode ser acompanhada por meio das operações de conexão e desconexão dos UEs. Um contador simples é usado para controlar o número de UEs conectados em cada célula. Vale destacar que um xApp seletor de células pode incluir diversas variáveis adicionais na escolha da célula, tais como, a taxa de transferência dos UEs conectados, a interferência dentro da célula e a potência total de transmissão e recepção. Entretanto, para o propósito da avaliação da RFT estas métricas não são realmente necessárias, pois apenas o cálculo do critério de seleção de células seria alterado.

A RFT utiliza replicação parcial e o estado do *Cell Selector xApp* é particionado em contextos, cada um representando uma determinada célula. No experimento, cada réplica do xApp acompanha a utilização de um conjunto células, diferentes réplicas do xApp acompanham

diferentes conjuntos. Assim, cada réplica mantém informações atualizadas sobre o grupo correspondente de células. A biblioteca RFT foi utilizada para gerenciar a replicação dos contextos entre cada réplica primária do xApp e outra réplica *backup*.

Os experimentos compararam um xApp implementado com a RFT (que armazena os contextos localmente) com o mesmo xApp implementado com uma solução baseada no Redis (Redislabs, 2020) (que mantém os contextos em uma base de dados externa). O Redis pode ser usado como um banco de dados e permite o armazenamento de estruturas de dados baseadas em *chave-valor* em memória principal. A escolha do Redis foi considerada pelo fato de que ele normalmente fornece armazenamento de alto desempenho e baixa latência, suporta alta disponibilidade e replicação, possibilita o particionamento automático de estado (*e.g.*, *sharding*) e também é utilizado como base para a implementação da SDL no RIC.

5.7.1 Configuração dos Experimentos

Os experimentos foram realizados em ambiente de laboratório no qual os xApps foram executados em contêineres divididos em duas máquinas físicas chamadas de *Server1* e *Server2* conectadas a uma rede Gigabit. A máquina *Server1* executou Linux Ubuntu 18.04 em um processador Intel(R) Core(TM) i7-6700HQ @ 2.6 GHz com 4 núcleos, 12 GiB de memória RAM e 6144K de cache L3. A máquina *Server2* executou Linux Ubuntu 16.04 em um processador Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz com 4 núcleos, 8 GiB de memória RAM e 8192K cache L3. Cinco contêineres foram executados no experimento da RFT, dois no *Server1* e três no *Server2*. Cada contêiner executou uma única réplica do xApp. As réplicas *backup* da RFT foram selecionadas conforme descrito anteriormente na Seção 5.4.

Cinco réplicas do xApp também foram utilizadas no experimento com o Redis e foram divididas da mesma forma que no experimento da RFT (dois xApps/contêineres no *Server1* e três xApps/contêineres no *Server2*). No experimento com o Redis, cada réplica do xApp armazena o seu estado (*i.e.*, contextos) em uma determinada réplica primária do Redis (*master*). Cada réplica primária do Redis executa localmente na mesma máquina da respectiva réplica do xApp. Além das cinco réplicas primárias do Redis, foram executadas outras 5 réplicas *backup* do Redis com o mesmo cenário de réplicas *backup* usado no experimento com a RFT.

São necessárias cinco réplicas adicionais de *backup* para o Redis, uma vez que o mesmo não permite que uma mesma instância execute como réplica primária e *backup* simultaneamente. A RFT permite e implementa essa funcionalidade. A configuração de uma instância *backup* para cada réplica primária do Redis foi realizada manualmente, caso contrário seria necessário executar serviços adicionais do Redis gerando maior sobrecarga de processamento. A distribuição de carga (*i.e.*, três contextos por réplica do xApp e por instância primária do Redis) é a mesma usada no experimento com a RFT. Vale ressaltar que foram necessários 5 contêineres para executar o experimento da RFT, enquanto que no experimento com o Redis foram necessários 5 contêineres para as réplicas do xApp e mais 10 contêineres executando o Redis para prover uma solução de replicação equivalente. O xApp que utiliza a solução do Redis para armazenar o estado foi implementado utilizando comunicação assíncrona com a réplica primária do Redis. A API cliente oficial do Redis foi utilizada nesta implementação (Redislabs, 2020).

Também foi implementado um gerador de tráfego na linguagem C para mensurar e comparar a abordagem da RFT com a abordagem utilizando o Redis em termos de latência e vazão da rede. Foram executadas cinco instâncias do gerador de tráfego, cada uma delas na mesma máquina física da respectiva réplica primária do xApp. Cada gerador de tráfego troca mensagens usando o RMR com a réplica correspondente do xApp e implementa a combinação das funcionalidades de um elemento da RAN e de um componente *E2 Term* da plataforma RIC. A vazão e latência foram mensuradas pelo gerador de tráfego. Além disso, a latência foi mensurada

nos instantes de envio e recebimento das respectivas mensagens. O experimento foi repetido 50 vezes enviando 100.000 mensagens em cada rodada. Cada mensagem foi enviada em: (i) intervalos de 1 microssegundo; e, (ii) vazão máxima suportada pelo RMR em cada máquina.

5.7.2 Resultados Experimentais

Inicialmente foi avaliada a vazão do *Cell Selector xApp* e os resultados são mostrados na Figura 5.9. As réplicas 1 e 2 do xApp foram executadas no *Server1* e as réplicas 3 a 5 foram executadas no *Server2*. O motivo para executar 2 réplicas no *Server1* e 3 réplicas no *Server2* é o fato de que o *Server2* possui uma CPU melhor. Uma única instância do xApp (não replicada) e uma instância do gerador de tráfego foram consideradas na medição de base (*baseline*) deste experimento. É possível observar que a vazão foi em média próxima a 125.000 mensagens por segundo para a RFT. O xApp equivalente implementado usando Redis atingiu em média uma vazão de pouco mais de 48.000 mensagens por segundo. É importante lembrar que o Redis não foi configurado para armazenar os dados no disco. Contudo, o xApp necessitou acessar as informações de contexto no contêiner Redis que estava executando na mesma máquina que a respectiva instância do xApp.

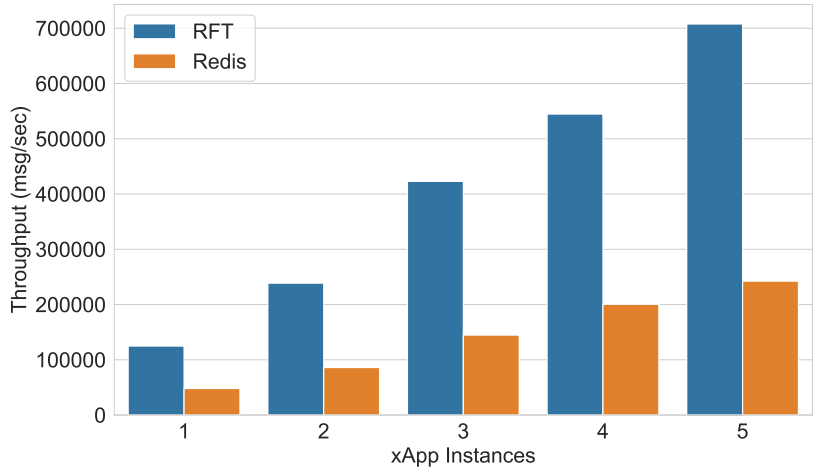


Figura 5.9: Comparação entre a vazão do *Cell Selector xApp* utilizando a RFT e o Redis.

Também foi mensurada a vazão das abordagens RFT e Redis considerando a replicação parcial de estados. É possível perceber na Figura 5.9 que ambas as abordagens são capazes de aumentar a vazão na medida em que o número de réplicas aumenta. Ainda pode-se perceber que houve um aumento maior na vazão ao adicionar a terceira réplica (que executa no *Server2*) se comparada com a adição da segunda réplica (que executa no *Server1*). O motivo deve-se ao fato de que o *Server2* executa em hardware melhor comparado com o *Server1*. Esse comportamento ocorre tanto para o experimento com a RFT quanto para o Redis. Além disso, ao comparar a vazão da RFT empregando a replicação parcial de estados com vazão da RFT sem replicação (*i.e.*, 1 instância), pode-se verificar que a taxa de transferência da RFT aumenta em 91,16%, 238,4%, 336,42% e 467% conforme o número de réplicas do xApp aumenta respectivamente para 2, 3, 4 e 5.

Em seguida foi executado um experimento para avaliar a latência de ida e volta (*i.e.*, *round-trip*) percebida pelo componente *E2 Term* da plataforma RIC que é implementado nos experimentos pelo gerador de tráfego. Foi calculado o intervalo de tempo desde o envio de uma mensagem do gerador de tráfego para a réplica correspondente do xApp até a chegada da

respectiva resposta. Inicialmente, a latência foi mensurada com o gerador de tráfego enviando mensagens em um ritmo lento: uma única mensagem foi enviada a cada microssegundo. A Figura 5.10 mostra a latência média com o intervalo de confiança de 95% calculado. Foram enviadas 100.000 mensagens em cada execução do experimento que foi repetido por 50 vezes.

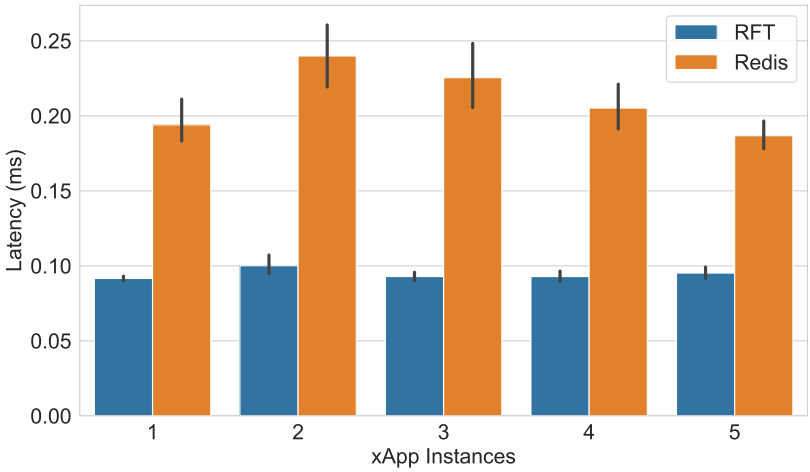


Figura 5.10: Latência média do *Cell Selector xApp* com carga de 1 mensagem por microssegundo.

É possível notar na Figura 5.10 que a RFT apresentou latência média inferior ao Redis neste experimento (RFT: 100 microssegundos & Redis: 240 microssegundos), porém ambas as latências podem ser consideradas baixas o suficiente para os requisitos do RIC. Os xApps que utilizam o Redis são mais lentos, pois a troca de mensagens na rede impõe maior latência mesmo executando os contêineres no mesmo hardware. A Figura 5.10 também mostra uma redução na latência média quando as réplicas que executam no *Server2* – que tem melhor CPU – são adicionadas ao experimento. No próximo experimento, são mostrados os resultados obtidos ao aumentar quantidade de requisições por segundo.

A Figura 5.11 ilustra a latência média obtida ao aumentar a taxa de requisições (*i.e.*, vazão) ao máximo suportado pela combinação do gerador de tráfego e do RMR no *Server1* e *Server2*. Os resultados apresentados são as médias de 50 execuções de 100.000 mensagens com o intervalo de confiança de 95%. Conforme o resultado mostrado nessa figura, a RFT foi capaz de manter baixa latência conforme o número de réplicas aumenta. A média da latência calculada para uma única instância do xApp foi de 0,944ms. A latência média aumenta para 1,2ms na medida em que são executadas 5 réplicas do xApp.

Ao executar o experimento usando a mesma taxa de requisições utilizando a abordagem com o Redis, a latência atingiu em média 80ms mesmo ao executar uma única instância do xApp e uma réplica do Redis. A latência significativamente maior mostrada na Figura 5.11 deve-se ao fato de que o Redis não foi capaz de sustentar a vazão de 125.000 requisições por segundo por réplica de xApp. Na abordagem implementada com o Redis, a réplica do xApp deve aguardar a resposta do Redis para então enviar a mensagem de resposta ao gerador de tráfego, o que causa o aumento da latência e diminui a taxa de transferência em comparação com a RFT. Além disso, os experimentos de latência e vazão apresentados anteriormente referem-se às mesmas execuções. Os dados calculados para a latência foram obtidos do mesmo conjunto de dados usados para calcular a vazão. Dessa forma, é possível comparar o comportamento da vazão e da latência na medida em que o número de réplicas aumenta.

Os resultados dos experimentos confirmam que a abordagem da RFT atende aos requisitos de latência do RIC (em média abaixo de 2ms) e é capaz de sustentar vazão escalável

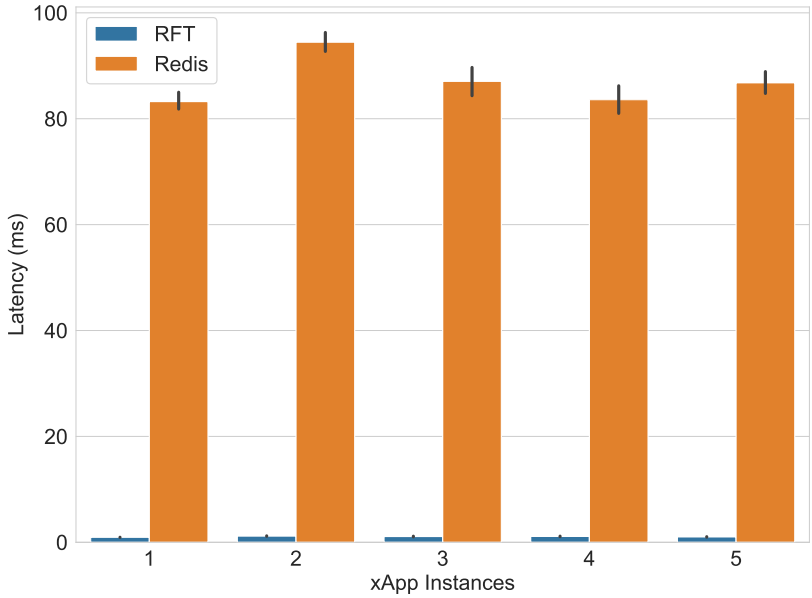


Figura 5.11: Latência média para a vazão máxima de requisições suportada.

para processar centenas de milhares de requisições por segundo. Experimentos específicos de tolerância a falhas não puderam ser executados devido a algumas limitações do controlador RIC em seu estágio atual de desenvolvimento. Foram identificados e reportados diversos problemas (*bugs*) no desenvolvimento do RIC durante e após o estágio sanduíche. Uma contribuição recente deste trabalho de doutorado foi correção de um erro que impedia o RMR entregar mensagens *multicast* para as réplicas do grupo de um xApp em determinadas situações. Este erro afetava a eleição de líder da RFT e também componentes da própria plataforma RIC. A seguir, são apresentados os trabalhos relacionados no contexto de tolerância a falhas.

5.8 TRABALHOS RELACIONADOS

O problema de fornecer tolerância a falhas e ao mesmo tempo garantir baixa latência e alta vazão foi abordado de diferentes maneiras ao longo dos anos. A tolerância a falhas é tipicamente alcançada usando replicação. Por exemplo, a LLFT (*Low Latency Fault Tolerance*) (Zhao et al., 2013) é baseada em replicação com líder primário. O sistema garante a consistência forte da réplica e baixa sobrecarga, pois a réplica primária pode responder sem aguardar pelas respostas das réplicas *backup*, mas a ordem das mensagens estabelecida pela réplica primária garante que as outras réplicas atingirão o mesmo estado. Embora o sistema forneça baixa latência, a vazão é limitada, pois o estado é totalmente replicado e o sistema não pode utilizar as diferentes réplicas para tarefas distintas simultâneas. Na RFT, o estado é particionado e parcialmente replicado permitindo que o processamento das requisições seja realizado em paralelo, no sentido de que as requisições são encaminhadas para as réplicas que possuem as informações correspondentes a cada requisição.

A replicação em cadeia (*chain replication*) (van Renesse e Schneider, 2004) atinge alta disponibilidade com a replicação de um objeto em t réplicas, sendo que t-1 réplicas podem falhar sem comprometer a disponibilidade do objeto replicado. Enquanto que as operações de leitura dos objetos são executadas na primeira réplica da cadeia, as atualizações são processadas pela última réplica da cadeia. A consistência é garantida pelo fato de que a alteração de estado deve passar por todas as réplicas até chegar à última da cadeia. O desempenho da leitura é beneficiado,

pois essas operações envolvem apenas uma única réplica, enquanto que atualizações envolvem o processamento por todas as réplicas, e a latência pode crescer significativamente mesmo para poucas réplicas.

O sistema descrito em Sherry et al. (2015) apresenta uma solução de alta disponibilidade para *middleboxes* usando máquinas virtuais. A estratégia assume que os *middleboxes* mantêm estado que deve ser devidamente recuperado após uma falha. O sistema é baseado na estratégia clássica de *rollback recovery*. Para aumentar o desempenho é empregada uma estratégia de *logging*. Máquinas virtuais executam uma réplica primária e outra de *backup* enquanto que dois registradores são utilizados para armazenar o tráfego de entrada e saída em um *switch* virtual. *Checkpoints* periódicos da réplica primária são salvos na memória principal do *switch* que são utilizados para inicializar o estado da réplica de *backup* caso a primária falhe. Os pacotes da rede somente são liberados pelo *switch* após todas as informações necessárias para a retransmissão do pacote terem sido armazenadas. Além disso, a função de rede implementada no *middlebox* não é executada enquanto um *checkpoint* está sendo processado. Portanto, o impacto na latência é proporcional à quantidade de alterações do estado da aplicação entre os *checkpoints* e é inversamente proporcional à largura de banda exigida pelo armazenamento.

Uma alternativa para atingir replicação com desempenho escalável é permitir que diferentes réplicas processem diferentes subconjuntos de mensagens e fragmentem o armazenamento do estado da aplicação. Na área de banco de dados, a técnica de *sharding* permite a divisão de estado em diferentes réplicas (Cattell, 2011). Essas divisões de estado, denominadas *shards*, podem ser replicadas tantas vezes quanto necessário e podem ser configuradas para processarem um maior número de leituras ou atualizações simultâneas.

Em Adya et al. (2016), é proposta uma estratégia que permite que aplicações de *datacenter* distribuam a carga de trabalho por meio de um serviço de fragmentação denominado Slicer. No Slicer, uma tarefa é um processo de uma aplicação que executa de forma simultânea com processos de outras aplicações em um hardware compartilhado. O Slicer utiliza, além da fragmentação, o balanceamento de carga na execução das tarefas. O sistema é monitorado para detectar congestionamento e falhas. O Slicer tem como maior objetivo realizar o balanceamento de carga uniforme. A fragmentação da carga de trabalho é feita utilizando funções *hash*. O Slicer armazena o estado das aplicações em um sistema externo, como o Redis.

Com relação aos esforços da O-RAN em geral, as iniciativas OAI (*Open Air Interface*) e srsLTE visam fornecer implementações de código aberto dos padrões 3GPP para a RAN (Gomez-Miguelez et al., 2016; Kaltenberger et al., 2019; OAI, 2020; O-RAN SC, 2020b). Não foram encontrados trabalhos que abordam a tolerância a falhas.

5.9 CONCLUSÃO

Este capítulo apresentou a estratégia para a construção de microsserviços tolerantes a falhas RFT (*RIC Fault Tolerance*) que visa resolver o problema de tornar um controlador O-RAN altamente disponível e que, simultaneamente, apresente alta vazão e baixa latência – abaixo de 2ms. A solução proposta baseia-se no particionamento de estados com replicação parcial aproximada em grupos de xApps e re-roteamento de mensagens com ciência de papel. As réplicas primárias replicam as atualizações dos contextos para as réplicas *backup*. As réplicas *backup* processam as mensagens com a ciência de que são *backup* e podem executar um comportamento diferente da réplica primária. A RFT foi implementada e disponibilizada como uma biblioteca que possibilita implementar xApps tolerantes a falhas. Resultados experimentais mostram que a RFT foi capaz de atender aos requisitos de latência e ao mesmo tempo sustentar vazão de forma escalável para processar centenas de milhares de requisições por segundo.

6 CONCLUSÃO

Esta Tese de Doutorado apresentou contribuições em dois contextos diferentes de pesquisa: a composição e execução de serviços virtualizados de rede sobre múltiplos domínios e orquestradores NFV; e a tolerância a falhas para microsserviços do controlador O-RAN.

No contexto da NFV, foi primeiro proposto o *Holistic-Composer* para permitir a composição e o gerenciamento do ciclo de vida de uma SFC que pode ser orquestrada por diferentes plataformas NFV. O gerenciamento do ciclo de vida compreende operações de instanciação, monitoramento, encerramento e alocação dos recursos necessários para a execução da SFC. Na prática, o *Holistic-Composer* resolve um problema desafiador, pois a composição e o gerenciamento do ciclo de vida de uma SFC diferem significativamente entre as diversas plataformas de orquestração NFV existentes, envolvendo a execução de tarefas detalhadas e complexas pelos operadores de rede e exigindo conhecimentos específicos de operação de diversas plataformas existentes. O *Holistic-Composer* define uma abordagem holística que permite abstrair as configurações de baixo nível necessárias para a composição e execução da SFC em diferentes orquestradores NFV. Uma API genérica e única oferece uma abstração genérica para a composição e orquestração de SFCs. Agentes de comunicação são utilizados para traduzir as operações genéricas fornecidas pela API para as operações específicas dos orquestradores NFV correspondentes. A solução é extensível e baseada nos padrões NFV-MANO da ETSI.

Um protótipo do *Holistic-Composer* foi implementado como prova de conceito para realizar a composição e o gerenciamento do ciclo de vida de VNFs que executam o Click-on-OSv sobre a plataforma de orquestração NFV Tacker. Experimentos foram executados e permitiram avaliar a proposta de forma quantitativa e qualitativa. Na avaliação quantitativa pôde-se verificar que foram necessários em média 17,5ms para adicionar uma nova VNF na SFC executando um único cliente, enquanto que foram necessários em média 251ms para compor uma SFC com 10 VNFs. Além disso, a composição de 128 SFCs de forma concorrente foi 4,145 vezes mais rápida em comparação com a versão sequencial. Foram necessários em média pouco mais de 6 segundos para compor 128 SFCs com tamanho de 10 VNFs de forma concorrente. A avaliação qualitativa permitiu verificar que o *Holistic-Composer* é capaz de compor toda a SFC apenas com a informação de sua sequência de VNFs, possibilitando abstrair as particularidades do orquestrador NFV utilizado.

Como segunda contribuição da Tese no contexto de NFV, foi apresentada a Multi-SFC: uma abordagem holística para a composição e execução de SFCs em múltiplas nuvens, orquestradas por diferentes plataformas NFV, e em múltiplos domínios, que podem ser federados. Uma Multi-SFC é formada por blocos básicos denominados segmentos. Cada segmento agrupa todas as VNFs que estão conectadas em uma determinada nuvem/domínio/orquestrador. Segmentos são interconectados por túneis implementados como VNFs, que passam a fazer parte da composição. Uma API genérica foi definida, a qual abstrai as particularidades dos diferentes orquestradores NFV para a composição e execução de SFCs que atravessam múltiplos domínios, inclusive federados. Pela sua flexibilidade na composição de SFCs, a Multi-SFC permite que operadores de rede escolham a plataforma/domínio mais adequadas para executar as VNFs de uma SFC, usando critérios como custo, desempenho e localização. Abordagens atuais restringem os operadores de rede a compor as suas SFCs utilizando serviços de rede pré-existentes e providos diretamente pelas interfaces dos orquestradores NFV. A Multi-SFC adota uma abordagem mais ampla, na qual os domínios que participam do acordo de federação cooperam com o compartilhamento das plataformas de nuvem e orquestração subjacentes. Assim,

a execução de uma VNF se torna independente de domínio e possibilita que composições de SFCs formem serviços de rede distintos.

Embora a arquitetura da Multi-SFC esteja alinhada com a especificação NFV-MANO da ETSI, a Multi-SFC é flexível no sentido de que ela também pode ser utilizada com plataformas NFV que não são totalmente compatíveis com o NFV-MANO. Essencialmente, as implementações dos respectivos *drivers* para os módulos *VIM Drivers* e *NFVO Drivers* devem disponibilizar operações básicas do ciclo de vida de VNFs e SFCs (*i.e.*, composição, instanciação, execução e encerramento) e não dependem exclusivamente do NFV-MANO. Em particular, após uma investigação cuidadosa da literatura da área, é possível afirmar que a Multi-SFC é a primeira estratégia existente para a construção de uma SFC que permite que mais de um orquestrador NFV seja utilizado para compor uma SFC em uma federação. Um protótipo da Multi-SFC foi implementado como prova de conceito utilizando os orquestradores NFV Tacker e OSM e a plataforma de computação em nuvem OpenStack. Resultados experimentais mostraram que a Multi-SFC apresenta baixa latência e mantém vazão compatível com a implementação do túnel e o hardware utilizado. A sobrecarga apresentada pelos túneis está relacionada à tecnologia utilizada e difere conforme o serviço que cada um deles fornece (*e.g.*, criptografia).

Com base nas evidências apresentadas nesta Tese de Doutorado é possível confirmar a **Hipótese 1** que foi apresentada no Capítulo 1. Dado o cenário apresentado no Capítulo 4 pode-se afirmar que é possível abstrair as especificidades de diferentes plataformas e orquestradores NFV para compor e implantar SFCs através de uma API genérica e única. A abordagem holística empregada pelo *Holistic-Composer* e a definição da API genérica para a Multi-SFC permitiram abstrair as particularidades das plataformas de orquestração NFV Tacker e OSM, bem como da plataforma de nuvem OpenStack. Além disso, a implementação da Multi-SFC e os experimentos realizados permitiram confirmar que é possível compor e executar uma SFC em múltiplos domínios orquestrados por diferentes plataformas NFV sem a intervenção manual dos operadores de rede de cada domínio. Exemplos de composição descritos no Apêndice A corroboram esta afirmação. Também pode-se afirmar que um usuário com identidade federada é capaz de compor e executar uma SFC de forma flexível utilizando VNFs que não dependem dos serviços de rede providos pelas interfaces dos orquestradores NFV de cada domínio. A Multi-SFC representa uma abordagem mais genérica do que outras abordagens relacionadas existentes.

A terceira contribuição apresentada na Tese vem no contexto da O-RAN: uma estratégia para a tolerância a falhas do controlador RIC baseada na replicação de xApps. A estratégia consiste na definição de um conjunto de técnicas para permitir a implementação de xApps tolerantes a falhas capazes de suportar os requisitos de baixa latência e alta vazão impostos pela RAN, sobretudo em redes 5G. Foi proposta a utilização de técnicas de particionamento de estados com replicação parcial em grupos de xApps e re-roteamento de mensagens com ciência de papel para atender aos requisitos exigidos pelo controlador RIC. As políticas de roteamento são definidas tendo em vista a composição atualizada do grupo de réplicas correspondente, possibilitando que as mensagens sejam entregues para a réplica adequada para processar a mensagem. Uma biblioteca para a construção de microsserviços tolerantes a falhas para o RIC foi implementada e disponibilizada, denominada RFT (*RIC Fault Tolerance*). A RFT foi avaliada para a criação de um xApp seletor de células. A RFT também foi comparada com uma solução alternativa baseada na base de dados Redis que armazena as informações de estado dos xApps em memória principal. Resultados experimentais mostraram que apenas o xApp replicado com a RFT foi capaz de manter os requisitos de latência do RIC e sustentar vazão escalável para processar um alto volume de requisições por segundo.

As evidências apresentadas nesta Tese de Doutorado também confirmam a **Hipótese 2** levantada no Capítulo 1. Pode-se afirmar que é possível construir xApps tolerantes a falhas capazes

de processar mensagens da RAN com a baixa latência e alta vazão requisitadas pelo controlador RIC. Os resultados experimentais da RFT apresentados no Capítulo 5 permitem fundamentar a conclusão anterior, uma vez que o xApp replicado com a RFT foi capaz de responder às requisições com latência média inferior a 2ms mesmo no cenário de vazão máxima suportada pelo RMR. Experimentos realizados empregando a estratégia da RFT também confirmaram que é possível aumentar a vazão de um xApp conforme novas réplicas são adicionadas ao mesmo grupo, permitindo processar centenas de milhares de requisições por segundo.

Trabalhos futuros no contexto da NFV incluem a investigação de estratégias que permitem a alocação eficiente de recursos na orquestração de Multi-SFCs, incluindo a elasticidade e migração de VNFs a fim de otimizar a instanciação e execução dos serviços de rede. Também está prevista a integração da Multi-SFC com a plataforma NFV CloudStack/Vines (Flauzino et al., 2020) por meio da implementação de um *NFVO Driver* para comunicar com o Vines e um *VIM Driver* para trocar informações com a plataforma de nuvem CloudStack (Apache, 2020a). O uso de protocolos como o NSH (*Network Service Header*) e o SRv6 (*Segment Routing v6*) para o encaminhamento do tráfego de uma Multi-SFC também será investigado. Trabalhos futuros ainda incluem a investigação de estratégias que possibilitem a composição e orquestração de Multi-SFCs empregando o modelo de especificação de topologias CUSTOM (*CUstom Service TOpology Model*) (Garcia et al., 2020). Abstrair a composição de Multi-SFCs com "intenções" (*i.e.*, *intent-based Multi-SFCs*) é outra direção promissora de pesquisa.

No contexto da O-RAN, trabalhos futuros incluem a investigação de técnicas para elasticidade dos xApps tolerantes a falhas da plataforma RIC. Também será investigada a atribuição e o gerenciamento dinâmico das réplicas primária e de *backup* para os elementos da RAN, empregando técnicas de balanceamento de carga e posicionamento de xApps (*i.e.*, *placement*). Pretende-se também melhorar a estratégia de inicialização dos membros do grupo de um xApp, que atualmente requer que o líder inicial do grupo seja indicado por variável de ambiente. É objetivo avaliar a RFT para outros xApps na medida em que sejam implementados e disponibilizados pela comunidade O-RAN. Pretende-se ainda avaliar a RFT utilizando xApps tolerantes a falhas em uma rede 5G real. Além disso, investigar estratégias que possibilitem estender a abordagem de tolerância a falhas proposta para a RFT, em particular as técnicas de roteamento com ciência de papel, para outras áreas se mostra um trabalho futuro bastante promissor. Por fim, a abordagem de replicação de xApps da RFT está sendo proposta para a O-RAN SC, visando torná-la estratégia oficial da plataforma RIC em um futuro próximo.

REFERÊNCIAS

3GPP (2020a). The 3rd generation partnership project (3GPP). https://www.3gpp.org/. Acessado em dezembro de 2020.

3GPP (2020b). About 3GPP: Generations of mobile systems. https://www.3gpp.org/about-3gpp. Acessado em dezembro de 2020.

Adya, A., Myers, D., Howell, J., Elson, J., Meek, C., Khemani, V., Fulger, S., Gu, P., Bhuvanagiri, L., Hunter, J., Peon, R., Kai, L., Shraer, A., Merchant, A. e Lev-Ari, K. (2016). Slicer: Auto-sharding for datacenter applications. Em *Proc. OSDI*, páginas 739–753.

Akman, A., Li, C., Ong, L., Suciu, L., Sahin, B. Y., Li, T., Stjernholm, P., Voigt, J., Buldorini, A., Sun, Q. e et al (2020). O-RAN use cases and deployment scenarios: Towards open and smart RAN. White Paper, O-RAN Alliance.

Alharbi, T., Aljuhani, A. e Liu, H. (2017). Holistic DDoS mitigation using NFV. Em *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, páginas 1–4.

Apache (2020a). Apache CloudStack: Open source cloud computing. https://cloudstack.apache.org/. Acessado em dezembro de 2020.

Apache (2020b). Apache Zookeeper. https://zookeeper.apache.org/. Acessado em dezembro de 2020.

Baggio, G., Francescon, A. e Fedrizzi, R. (2017). Multi-domain service orchestration with X-MANO. Em *2017 IEEE Conference on Network Softwarization (NetSoft)*, páginas 1–2, Bologna, Italy.

Bernardos, C. J., Contreras, L. M., Vaishnavi, I., Szabo, R., Li, X., Paolucci, F., Sgambelluri, A., Martini, B., Valcarenghi, L., Landi, G., Andrushko, D. e Mourad, A. (2019). Multi-domain network virtualization. Internet-Draft draft-bernardos-nmrg-multidomain-01, Internet Engineering Task Force. Work in Progress.

Bhardwaj, K., Miranda, J. C. e Gavrilovska, A. (2018). Towards IoT-DDoS prevention using edge computing. Em *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA. USENIX Association.

Bondan, L., Franco, M. F., Marcuzzo, L., Venancio, G., Santos, R. L., Pfitscher, R. J., Scheid, E. J., Stiller, B., Turck, F. D., Duarte, E. P., Schaeffer-Filho, A. E., Santos, C. R. P. d. e Granville, L. Z. (2019). FENDE: Marketplace-based distribution, execution, and life cycle management of VNFs. *IEEE Communications Magazine*, 57(1):13–19.

Cachin, C., Guerraoui, R. e Rodrigues, L. (2011). *Introduction to reliable and secure distributed programming*. Springer Science & Business Media.

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM SIGMOD Rec.*, 39(4):12–27.

Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Cui, C., Deng, H. e Michel, U. (2012). Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action. White paper, ETSI.

Chiosi, M., Wright, S., Clarke, D., Willis, P., Johnson, L., Bugenhagen, M., Feger, J., Khan, W., Chunfeng, C. e et al (2013). Network functions virtualisation (NFV): Network operator perspectives on industry progress. White paper, ETSI.

Ciena (2020). Blue planet multi-domain service orchestration (MDSO). `https://www.blueplanet.com/products/multi-domain-service-orchestration.html`. Acessado em dezembro de 2020.

Cloudify (2020). Network orchestration & edge networking. `https://cloudify.co/`. Acessado em dezembro de 2020.

Coletti, C., Diego, W., Duan, R., Ghassemzadeh, S., Gupta, D., Huang, J., Joshi, K., Matsukawa, R., Suciu, L., Sun, J. e et al (2018). O-RAN: Towards an open and smart RAN. White paper, O-RAN Alliance.

Cui, L., Tso, F. P. e Jia, W. (2020). Federated service chaining: Architecture and challenges. *IEEE Communications Magazine*, 58(3):47–53.

da Cruz Marcuzzo, L., Garcia, V. F., Cunha, V., Corujo, D., Barraca, J. P., Aguiar, R. L., Schaeffer-Filho, A. E., Granville, L. Z. e dos Santos, C. R. P. (2017). Click-on-OSv: A platform for running Click-based middleboxes. Em *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, páginas 885–886, Lisbon, Portugal.

de Sousa, N. F. S., Perez, D. A. L., Rosa, R. V., Santos, M. A. e Rothenberg, C. E. (2019). Network service orchestration: A survey. *Computer Communications*, 142-143:69–94.

Dorsemaine, B., Gaulier, J., Wary, J., Kheir, N. e Urien, P. (2015). Internet of things: A definition & taxonomy. Em *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, páginas 72–77.

ETSI (2014a). Network functions virtualisation (NFV); architectural framework. Group Specification ETSI GS NFV 002 V1.2.1, ETSI.

ETSI (2014b). Network functions virtualisation (NFV); terminology for main concepts in NFV. Group Specification GS NFV 003 V1.2.1, ETSI.

ETSI (2020a). Network functions virtualisation. `http://www.etsi.org/technologies-clusters/technologies/nfv`. Acessado em dezembro de 2020.

ETSI (2020b). Open source MANO. `https://osm.etsi.org/`. Acessado em dezembro de 2020.

Flauzino, J. W., Fülber-Garcia, V., de Souza, G. e Duarte Jr., E. P. (2020). Além do openstack: Disponibilizando o suporte para funções virtualizadas de rede NFV-MANO no cloudstack. Em *Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 435–448, Porto Alegre, RS, Brasil. SBC.

Fokus, F. e Tu, B. (2020). Open Baton: an open source reference implementation of the ETSI Network Function Virtualization MANO specification. `http://openbaton.github.io/`. Acessado em dezembro de 2020.

Francescon, A., Baggio, G., Fedrizzi, R., Orsini, E. e Riggio, R. (2017). X-MANO: An open-source platform for cross–domain management and orchestration. Em *2017 IEEE Conference on Network Softwarization (NetSoft)*, páginas 1–6, Bologna, Italy.

Garcia, V. F., Duarte, E. P., Huff, A. e dos Santos, C. R. (2020). Network service topology: Formalization, taxonomy and the CUSTOM specification model. *Computer Networks*, 178:107337.

Ghaznavi, M., Shahriar, N., Kamali, S., Ahmed, R. e Boutaba, R. (2017). Distributed service function chaining. *IEEE Journal on Selected Areas in Communications*, 35(11).

Gomez-Miguelez, I., Garcia-Saavedra, A., Sutton, P., Serrano, P., Cano, C. e Leith, D. (2016). srsLTE: an open-source platform for LTE evolution and experimentation. Em *Proc. 10th ACM Int. Workshop on Wireless Network Testbeds, Exp. Evaluation, and Characterization*, páginas 25–32.

GSMA (2020). The mobile economy 2020. `https://www.gsma.com/mobileeconomy/wp-content/uploads/2020/03/GSMA_MobileEconomy2020_Global.pdf`. Acessado em dezembro de 2020.

Gudipati, A., Perry, D., Li, L. E. e Katti, S. (2013). SoftRAN: Software defined radio access network. Em *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, página 25–30. ACM.

Gunleifsen, H., Kemmerich, T. e Gkioulos, V. (2019). Dynamic setup of IPsec VPNs in service function chaining. *Computer Networks*, 160:77 – 91.

Habibi, M. A., Nasimi, M., Han, B. e Schotten, H. D. (2019). A comprehensive survey of RAN architectures toward 5G mobile communication system. *IEEE Access*, 7:70371–70421.

Haitao, X. e Mann, A. (2018). Network functions virtualisation (NFV) release 3; management and orchestration; report on architecture options to support multiple administrative domains. Group Report GR NFV-IFA 028 V3.1.1, ETSI.

Haitao, X., Nicolas, A. M. D., Jie, M., Gang, H. e Khasnabish, B. (2018). Network functions virtualisation (NFV) release 3; management and orchestration; multiple administrative domain aspect interfaces specification. Group Specification ETSI GS NFV-IFA 030 V3.1.1, ETSI.

Halpern, J. M. e Pignataro, C. (2015). Service function chaining (SFC) architecture. RFC 7665.

Han, B., Gopalakrishnan, V., Ji, L. e Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

Heat (2020). Heat - openstack orchestration. `https://wiki.openstack.org/wiki/Heat`. Acessado em dezembro de 2020.

Huff, A., Hiltunen, M. e Duarte Jr., E. P. (2020). RFT: RIC fault tolerance. `https://github.com/alexandre-huff/rft`. Acessado em dezembro de 2020.

Huff, A., Venâncio, G., da C. Marcuzzo, L., Garcia, V. F., dos Santos, C. R. P. e Duarte Jr., E. P. (2018a). A holistic approach to define service chains using Click-on-OSv on different NFV platforms. Em *2018 IEEE Global Communications Conference (GLOBECOM)*, páginas 1–6, Abu Dhabi, UAE.

Huff, A., Venâncio, G., da C. Marcuzzo, L., Garcia, V. F., dos Santos, C. R. P. e Duarte Jr., E. P. (2018b). Uma abordagem holística para a definição de service chains utilizando Click-on-OSv sobre diferentes plataformas NFV. Em *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, volume 36, Porto Alegre, RS, Brasil. SBC.

Huff, A., Venâncio, G. e Duarte Jr., E. P. (2019). Multi-SFC: Orquestração de SFCs distribuídas sobre múltiplas nuvens em múltiplos domínios com múltiplas plataformas NFV. Em *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, volume 37, páginas 777–790, Porto Alegre, RS, Brasil. SBC.

Huff, A., Venâncio, G., Garcia, V. F. e Duarte Jr., E. P. (2020). Building multi-domain service function chains based on multiple NFV orchestrators. Em *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, páginas 19–24, Leganes - Madrid, Spain.

Huff, A., Venâncio, G., Garcia, V. F. e Duarte Jr., E. P. (2020). Multi-SFC. `https://github.com/alexandre-huff/multi-sfc`. Acessado em dezembro de 2020.

IETF (2020). Service function chaining (SFC) - documents. `https://datatracker.ietf.org/wg/sfc/documents/`. Acessado em outubro de 2020.

Joshi, K. D. e Kataoka, K. (2020). pSMART: A lightweight, privacy-aware service function chain orchestration in multi-domain NFV/SDN. *Computer Networks*, 178:107295.

Kaltenberger, F., de Souza, G., Knopp, R. e Wang, H. (2019). The OpenAirInterface 5G new radio implementation: Current status and roadmap. Em *23rd International ITG Workshop on Smart Antennas (WSA)*.

Keltoum, B. e Samia, B. (2017). A dynamic federated identity management approach for cloud-based environments. Em *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, Cambridge, United Kingdom. Association for Computing Machinery.

Kim, H. e Feamster, N. (2013). Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119.

Kohler, E., Morris, R., Chen, B., Jannotti, J. e Kaashoek, M. F. (2000). The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297.

Kojukhov, A., de Nicolas, A. M., Chatras, B., Druta, D., Gassanov, D., Brunner, M., Brenner, M., Li, S., Nguyenphu, T., Rauschenbach, U. e Sacks, Z. (2017). Network functions virtualisation (NFV) release 2; protocols and data models; VNF Package specification. Group Specification ETSI GS NFV-SOL 004 V2.3.1, ETSI.

Kubernetes (2020). Kubernetes. `https://kubernetes.io/`. Acessado em dezembro de 2020.

Laurent, M. e Bouzefrane, S. (2015). *Digital identity management*. Elsevier, London.

Li, S. e Crandall, J. (2017). TOSCA simple profile for network functions virtualization (NFV) version 1.0. OASIS committee specification draft, Open Standards for the Information Society. `http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html`.

Liang, L., Zheng, K., Sheng, Q. e Huang, X. (2016). A denial of service attack method for an iot system. Em *Proc. 8th Int. Conf. on Information Technology in Medicine and Education (ITME)*, páginas 360–364.

Linux Foundation (2020a). DPDK - data plane development kit. http://www.dpdk.org/. Acessado em dezembro de 2020.

Linux Foundation (2020b). Open vSwitch. http://www.openvswitch.org/. Acessado em dezembro de 2020.

Linux Foundation (2020c). OPNFV. https://www.opnfv.org/. Acessado em dezembro de 2020.

Lockhart, H., Campbell, B., Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P. e Scavo, T. (2008). Security assertion markup language (SAML) V2.0 technical overview. Relatório Técnico Committee Draft 02, OASIS.

Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R. e Huici, F. (2014). ClickOS and the art of network function virtualization. Em *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, páginas 459–473, Berkeley, CA, USA. USENIX Association.

Mell, P. e Grance, T. (2011). The NIST definition of cloud computing. https://www.nist.gov/publications/nist-definition-cloud-computing. Acessado em dezembro de 2020.

Memcached (2020). Memcached - a distributed memory object caching system. https://memcached.org/. Acessado em dezembro de 2020.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. e Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262.

O-RAN Alliance (2019). O-RAN Alliance. https://www.o-ran.org/. Acessado em dezembro de 2020.

O-RAN Alliance (2020a). O-RAN architecture description. Technical Specification O-RAN-WG1-O-RAN Architecture Description - v01.00.00, O-RAN Alliance.

O-RAN Alliance (2020b). O-RAN near-real-time RAN intelligent controller architecture & E2 general aspects and principles 1.01. Technical Specification O-RAN.WG3.E2GAP-v01.01, O-RAN Alliance.

O-RAN Alliance (2020c). O-RAN operations and maintenance architecture. Technical Specification O-RAN.WG1.OAM-Architecture-v03.00, O-RAN Alliance.

O-RAN SC (2020a). O-RAN SC projects: Near realtime RAN intelligent controller (RIC). https://docs.o-ran-sc.org/en/latest/projects.html. Acessado em dezembro de 2020.

O-RAN SC (2020b). O-RAN software community. https://o-ran-sc.org/. Acessado em dezembro de 2020.

OAI (2020). OpenAirInterface: 5G software alliance for democratising wireless innovation. https://www.openairinterface.org/. Acessado em dezembro de 2020.

Olwal, T. O., Djouani, K. e Kurien, A. M. (2016). A survey of resource management toward 5G radio access networks. *IEEE Communications Surveys & Tutorials*, 18(3):1656–1686.

ONAP (2020). ONAP - open network automation platform. https://www.onap.org/. Acessado em dezembro de 2020.

ONF (2020). SDN technical specifications. https://www.opennetworking.org/software-defined-standards/specifications/. Acessado em dezembro de 2020.

Ongaro, D. (2014). *Consensus: Bridging theory and practice*. PhD Thesis, Stanford University.

Ongaro, D. e Ousterhout, J. (2014). In search of an understandable consensus algorithm. Em *Proc. USENIX ATC*, páginas 305–319.

OpenStack (2020a). OpenStack - open source software for creating private and public clouds. https://www.openstack.org/. Acessado em dezembro de 2020.

OpenStack (2020b). Openstack docs: Keystone, the openstack identity service. https://docs.openstack.org/keystone/latest/. Acessado em dezembro de 2020.

OSv (2020). OSv - the operating system designed for the cloud. http://osv.io/. Acessado em dezembro de 2020.

Panda, A., Han, S., Jang, K., Walls, M., Ratnasamy, S. e Shenker, S. (2016). Netbricks: Taking the V out of NFV. Em *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, páginas 203–216, Savannah, GA.

Parvez, I., Rahmati, A., Guvenc, I., Sarwat, A. I. e Dai, H. (2018). A survey on low latency towards 5G: RAN, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3098–3130.

Quinn, P., Elzur, U. e Pignataro, C. (2018). Network service header (NSH). RFC 8300.

Quinn, P. e Nadeau, T. (2015). Problem statement for service function chaining. RFC 7498.

Quittek, J., Bauskar, P., BenMeriem, T., Bennett, A., Besson, M. e et al (2014). Network functions virtualisation (NFV); management and orchestration. Group Specification GS NFV-MAN 001 V1.1.1, ETSI.

Redislabs (2020). Redis. https://redis.io/. Acessado em dezembro de 2020.

Riera, J. F., Batallé, J., Bonnet, J., Días, M., McGrath, M., Petralia, G., Liberati, F., Giuseppi, A., Pietrabissa, A., Ceselli, A. et al. (2016). Tenor: Steps towards an orchestration platform for multi-PoP NFV deployment. Em *NetSoft Conf. and Workshops (NetSoft)*, páginas 243–250. IEEE.

Salsano, S., Lombardo, F., Pisa, C., Greto, P. e Blefari-Melazzi, N. (2017). RDCL 3D, a model agnostic web framework for the design and composition of NFV services. Em *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, páginas 216–222, Berlin, Germany.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, 22(4):299–319.

Sgambelluri, A., Tusa, F., Gharbaoui, M., Maini, E., Toka, L., Perez, J. M., Paolucci, F., Martini, B., Poe, W. Y., Melian Hernandes, J., Muhammed, A., Ramos, A., de Dios, O. G., Sonkoly, B., Monti, P., Vaishnavi, I., Bernardos, C. J. e Szabo, R. (2017). Orchestration of network services across multiple operators: The 5G exchange prototype. Em *European Conf. Networks and Communications (EuCNC)*, páginas 1–5.

Shayea, I., Ergen, M., Azmi, M. H., Çolak, S. A., Nordin, R. e Daradkeh, Y. I. (2020). Key challenges, drivers and solutions for mobility management in 5G networks: A survey. *IEEE Access*, 8:172534–172552.

Shen, W., Yoshida, M., Minato, K. e Imajuku, W. (2015). vConductor: An enabler for achieving virtual network integration as a service. *IEEE Communications Magazine*, 53(2):116–124.

Sherry, J., Gao, P. X., Basu, S., Panda, A., Krishnamurthy, A., Maciocco, C., Manesh, M., Martins, J. a., Ratnasamy, S., Rizzo, L. e et al. (2015). Rollback-recovery for middleboxes. Em *Proc. SIGCOMM*, página 227–240.

Sonkoly, B., Czentye, J., Szabo, R. et al. (2015). Multi-domain service orchestration over networks and clouds: a unified approach. *ACM SIGCOMM Comp. Comm. Review*, 45(4):377–378.

Tacker (2020). Tacker - openstack NFV orchestration. `https://docs.openstack.org/tacker/latest/`. Acessado em dezembro de 2020.

Trois, C., Fabro, D. D., D., M., de Bona, L. C. E. e Martinello, M. (2016). A survey on SDN programming languages: Towards a taxonomy. *IEEE Communications Surveys & Tutorials*, PP(99):1–1.

Uniyal, N., Muqaddas, A. S., Gkounis, D., Bravalheri, A., Moazzeni, S., Sardis, F., Dohler, M., Nejabati, R. e Simeonidou, D. (2020). 5GUK Exchange: Towards sustainable end-to-end multi-domain orchestration of softwarized 5G networks. *Computer Networks*, 178:107297.

van Renesse, R. e Schneider, F. B. (2004). Chain replication for supporting high throughput and availability. Em *Proc. OSDI*.

Yi, B., Wang, X., Li, K., Das, S. k. e Huang, M. (2018). A comprehensive survey of network function virtualization. *Computer Networks*, 133:212–262.

Yousaf, F. Z., Bredel, M., Schaller, S. e Schneider, F. (2017). NFV and SDN – key technology enablers for 5G networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478.

Yousaf, Z., Dempo, H. e Dietz, T. (2020). Network functions virtualisation (NFV) release 4; architectural framework report on network connectivity integration and operationalization for NFV. Group Report DGR/NFV-IFA035 v0.4.0, ETSI. Work in Progress.

Zayas, A. D. e Merino, P. (2017). The 3GPP NB-IoT system architecture for the internet of things. Em *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, páginas 277–282.

Zhang, Q., Cheng, L. e Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

Zhao, W., Melliar-Smith, P. e Moser, L. (2013). Low latency fault tolerance system. *The Computer Journal*, 56:716–740.

APÊNDICE A – EXEMPLOS DE COMPOSIÇÃO DE MULTI-SFCS

Este apêndice apresenta dois exemplos de composição de Multi-SFCs utilizando uma aplicação cliente implementada em Python que consome a interface REST provida pelo *Multi-SFC Orquestrator* descrito no Capítulo 4.

O primeiro exemplo mostra a composição de uma Multi-SFC em dois domínios que são orquestrados por diferentes plataformas NFV. Assume-se que todos os pacotes VNFs foram adicionados previamente no *VNF Catalog* do *Multi-SFC Orchestrator*. A Figura A.1 ilustra o usuário da aplicação cliente iniciando a composição de uma Multi-SFC que escolhe a opção *7*. Em seguida, usuário informa o domínio no qual o primeiro segmento será instanciado. Neste exemplo, *Domain 1* é escolhido. Também pode-se verificar que a plataforma NFV identificada como *tacker@east* é responsável por orquestrar os recursos computacionais providos pelo VIM identificado como *Cloud1*. A VNF que implementa o túnel usado para interconectar o domínio *Domain 1* com os outros domínios implementa o protocolo *IPSec*.

```
1. Include VNF Package
2. Remove VNF Package
3. Show VNF Catalog
4. Instantiate VNF
5. Destroy VNF Instance
6. Show VNF Instances
7. Compose Multi-SFC
8. Destroy Multi-SFC
9. Show Multi-SFC Instances
0. Exit

> 7

+-----+----------+-------------------+----------+--------+
| SEQ |  Domain  | Platform Instance | VIM Name | Tunnel |
+-----+----------+-------------------+----------+--------+
|  1  | Domain 1 |    tacker@east    |  Cloud1  | IPSec  |
|  2  | Domain 2 |     osm@west      |  Cloud2  | IPSec  |
+-----+----------+-------------------+----------+--------+
Select a domain/platform to compose this Multi-SFC segment
SEQ > 1
```

Figura A.1: Composição de uma nova instância da Multi-SFC.

A Figura A.2 mostra o próximo passo que consiste em definir a sequência de VNFs (*i.e.*, pacotes de VNFs que serão instanciados) que farão parte da composição de um determinado SFP (*Service Function Path*) da Multi-SFC. Como o usuário escolheu o domínio *Domain 1* que é orquestrado pela plataforma Tacker, somente são apresentados os pacotes de VNF correspondentes a esta plataforma. Ainda pode-se perceber que o pacote VNF *Forwarder 1* possui restrições adicionais se comparado com os outros pacotes VNF. As restrições denotam que este pacote VNF somente pode ser instanciado pela plataforma NFV denominada *tacker@east* no domínio *Domain 1*, enquanto que os outros pacotes VNF podem ser instanciados em qualquer domínio e qualquer uma das plataformas NFV Tacker (caso houver outras). Isso possibilita que pacotes VNF que tenham restrições específicas de hardware ou determinadas políticas administrativas só possam ser instanciados por uma plataforma particular em um domínio específico. Neste exemplo, o usuário realiza o encadeamento dos pacotes VNF *Forwarder 1* e *Forwarder 2* respectivamente.

Após realizar a composição do primeiro segmento, o usuário adiciona o próximo segmento da Multi-SFC conforme mostrado na Figura A.3. Neste exemplo, o usuário seleciona a instância da plataforma *osm@west* no domínio *Domain 2*. Também é possível perceber que a API

```
+-----+--------------+----------+----------+---------------+
| SEQ |  VNF Desc.   | Platform |  Domain  | Plat Instance |
+-----+--------------+----------+----------+---------------+
|  1  |    Client    |  Tacker  |   ANY    |      ANY      |
|  2  | Forwarder 1  |  Tacker  | Domain 1 |  tacker@east  |
|  3  | Forwarder 2  |  Tacker  |   ANY    |      ANY      |
|  4  | IPSec Tunnel |  Tacker  |   ANY    |      ANY      |
+-----+--------------+----------+----------+---------------+
Add VNFs by their Catalog SEQ (0 when done)
SEQ > 2
VNF included successfully!
SEQ > 3
VNF included successfully!
SEQ > 0
```

Figura A.2: Encadeamento de VNFs no domínio *Domain 1*.

do *Multi-SFC Core* retorna apenas os domínios e plataformas NFV que suportam túneis IPSec, uma vez que a plataforma do domínio anterior executa túneis VNF que implementam IPSec.

```
Add more Multi-SFC segments?
1. Yes
2. No
> 1

+-----+----------+-------------------+----------+--------+
| SEQ |  Domain  | Platform Instance | VIM Name | Tunnel |
+-----+----------+-------------------+----------+--------+
|  1  | Domain 1 |    tacker@east    |  Cloud1  | IPSec  |
|  2  | Domain 2 |     osm@west      |  Cloud2  | IPSec  |
+-----+----------+-------------------+----------+--------+
Select a domain/platform to compose this Multi-SFC segment
SEQ > 2
```

Figura A.3: Inclusão do próximo segmento da Multi-SFC.

Em seguida, uma lista de pacotes VNF é retornada à aplicação cliente, conforme mostrado na Figura A.4. O *Multi-SFC Core* retorna apenas os pacotes de VNF que podem ser instanciados pela plataforma OSM, uma vez que o domínio *Domain 2* é orquestrado pela plataforma NFV *osm@west*. Pode-se perceber ainda que o pacote VNF *HTTP Server* só pode ser executado pela instância da plataforma *osm@west* no domínio *Domain 2*, enquanto os outros pacotes VNF podem ser executados em qualquer domínio e qualquer plataforma NFV OSM. O usuário da aplicação cliente então encadeia os pacotes VNF *Forwarder 3* e *Forwarder 4* ao segmento da Multi-SFC no segundo domínio.

```
+-----+--------------+----------+----------+---------------+
| SEQ |  VNF Desc.   | Platform |  Domain  | Plat Instance |
+-----+--------------+----------+----------+---------------+
|  1  | HTTP Server  |   OSM    | Domain 2 |   osm@west    |
|  2  | Forwarder 3  |   OSM    |   ANY    |      ANY      |
|  3  | Forwarder 4  |   OSM    |   ANY    |      ANY      |
|  4  | IPSec Tunnel |   OSM    |   ANY    |      ANY      |
+-----+--------------+----------+----------+---------------+
Add VNFs by their Catalog SEQ (0 when done)
SEQ > 2
VNF included successfully!
SEQ > 3
VNF included successfully!
SEQ > 0
```

Figura A.4: Encadeamento de VNFs no domínio *Domain 2*.

As políticas do classificador de tráfego da Multi-SFC são configuradas após o usuário finalizar a composição dos segmentos de um SFP. A Figura A.5 mostra o usuário informando que o tráfego de origem da Multi-SFC será gerado por uma VNF da rede interna (*Internal*) que

executa na plataforma NFV do primeiro segmento. Na sequência, todas as VNFs instanciadas na plataforma NFV do primeiro segmento são apresentadas ao usuário. O usuário então seleciona a instância de VNF denominada *linux-client* como a origem do tráfego da Multi-SFC. Caso o usuário informe que o tráfego de origem da Multi-SFC será gerado a partir de uma rede externa (*External*), o *Multi-SFC Core* configura a interface de rede do roteador virtual como a origem do tráfego para a Multi-SFC. Outro exemplo de composição de uma Multi-SFC que recebe o tráfego externo é apresentado mais a frente.

```
Add more Multi-SFC segments?
1. Yes
2. No
> 2

Source of the Multi-SFC network traffic at the first segment
1. Internal
2. External
> 1

+-----+---------------+---------------+----------+--------+
| SEQ | Instance Name |    Address    | VIM Name | Status |
+-----+---------------+---------------+----------+--------+
|  1  | linux-client  | 192.168.120.9 |  Cloud1  | ACTIVE |
+-----+---------------+---------------+----------+--------+
Choose a VNF that generates the incoming Multi-SFC traffic
SEQ > 1
```

Figura A.5: Escolha da origem do tráfego da Multi-SFC.

A Figura A.6 mostra as políticas de classificação do tráfego da rede disponíveis para o primeiro segmento da Multi-SFC. As políticas de classificação dos segmentos seguintes são mapeadas e configuradas conforme as políticas correspondentes do primeiro segmento da Multi-SFC. Este exemplo mostra as políticas de classificação disponíveis na plataforma Tacker que executa no domínio *Domain 1*. O usuário então informa os critérios de classificação para o tráfego de entrada da Multi-SFC – neste exemplo, protocolo TCP (*6*), porta *80* e endereço de destino *10.10.0.12/32*. A Multi-SFC então é identificada por um nome (*example1*) e, por fim, a Multi-SFC é instanciada nas plataformas correspondentes.

É possível perceber no exemplo apresentado que nenhuma informação relacionada à configuração dos túneis VNF, roteamento entre domínios e restrições de segurança (*i.e.*, *firewall*) é inserida explicitamente pelo usuário durante a composição da Multi-SFC. O *Multi-SFC Core* é responsável por realizar essas configurações de forma transparente para o usuário com base nas informações registradas no *Domain Catalog* em conjunto com as políticas definidas no classificador de tráfego da Multi-SFC. Neste exemplo, os túneis IPSec são encadeados e configurados no final do primeiro segmento e no início do segundo segmento; as regras de roteamento entre as duas sub-redes são configuradas nos túneis VNF; e, os *firewalls* de ambas as infraestruturas de nuvem são configurados para aceitar pacotes TCP (*6*) na porta *80* com o endereço de destino IPv4 *10.10.0.12/32*.

```
+----------------+-------------------------------+
|       ID       |          Description          |
+----------------+-------------------------------+
|     arp_op     |          ARP opcode           |
|    arp_sha     |  ARP source hardware address  |
|    arp_spa     |    ARP source ipv4 address    |
|    arp_tha     |  ARP target hardware address  |
|    arp_tpa     |   ARP target ipv4 address     |
| dst_port_range |      Target port range        |
|    eth_dst     | Ethernet destination address  |
|    eth_src     |    Ethernet source address    |
|    eth_type    |      Ethernet frame type      |
|  icmpv4_code   |          ICMP code            |
|  icmpv4_type   |          ICMP type            |
|  icmpv6_code   |         ICMPv6 code           |
|  icmpv6_type   |         ICMPv6 type           |
| ip_dst_prefix  | IP destination address prefix |
|   ip_proto     |     IP protocol number        |
| ip_src_prefix  |   IP source address prefix    |
|   ipv6_dst     |   IPv6 destination address    |
|  ipv6_flabel   |      IPv6 Flow Label          |
|   ipv6_src     |    IPv6 source address        |
|  mpls_label    |         MPLS Label            |
|   vlan_id      |           VLAN ID             |
+----------------+-------------------------------+
Add the criteria by their respective ID (0 when done)
ID > ip_proto
Value > 6
ID > dst_port_range
Value > 80-80
ID > ip_dst_prefix
Value > 10.10.0.12/32
ID > 0

Define a name for this Multi-SFC instance (optional)
Name > example1

Multi-SFC example1 successfully instantiated!
```

Figura A.6: Definição dos critérios de classificação e instanciação da Multi-SFC.

Em seguida, é apresentado um exemplo de composição de uma Multi-SFC com a configuração de tráfego externo.

```
1. Include VNF Package
2. Remove VNF Package
3. Show VNF Catalog
4. Instantiate VNF
5. Destroy VNF Instance
6. Show VNF Instances
7. Compose Multi-SFC
8. Destroy Multi-SFC
9. Show Multi-SFC Instances
0. Exit

> 7

+-----+----------+--------------------+----------+--------+
| SEQ |  Domain  | Platform Instance | VIM Name | Tunnel |
+-----+----------+--------------------+----------+--------+
|  1  | Domain 1 |    tacker@east     |  Cloud1  | IPSec  |
|  2  | Domain 2 |     osm@west       |  Cloud2  | IPSec  |
+-----+----------+--------------------+----------+--------+
Select a domain/platform to compose this Multi-SFC segment
SEQ > 1

+-----+--------------+----------+----------+----------------+
| SEQ |  VNF Desc.   | Platform |  Domain  | Plat Instance |
+-----+--------------+----------+----------+----------------+
|  1  |    Client    |  Tacker  |   ANY    |      ANY       |
|  2  | Forwarder 1  |  Tacker  | Domain 1 |  tacker@east   |
|  3  | Forwarder 2  |  Tacker  |   ANY    |      ANY       |
```

```
|  4  | IPSec Tunnel |  Tacker  |   ANY    |      ANY      |
+-----+--------------+----------+----------+---------------+
Add VNFs by their Catalog SEQ (0 when done)
SEQ > 2
VNF included successfully!
SEQ > 3
VNF included successfully!
SEQ > 0

Add more Multi-SFC segments?
1. Yes
2. No
> 1

+-----+----------+-------------------+----------+--------+
| SEQ |  Domain  | Platform Instance | VIM Name | Tunnel |
+-----+----------+-------------------+----------+--------+
|  1  | Domain 1 |    tacker@east    |  Cloud1  | IPSec  |
|  2  | Domain 2 |     osm@west      |  Cloud2  | IPSec  |
+-----+----------+-------------------+----------+--------+
Select a domain/platform to compose this Multi-SFC segment
SEQ > 2

+-----+--------------+----------+----------+---------------+
| SEQ |  VNF Desc.   | Platform |  Domain  | Plat Instance |
+-----+--------------+----------+----------+---------------+
|  1  | HTTP Server  |   OSM    | Domain 2 |   osm@west    |
|  2  | Forwarder 3  |   OSM    |   ANY    |      ANY      |
|  3  | Forwarder 4  |   OSM    |   ANY    |      ANY      |
|  4  | IPSec Tunnel |   OSM    |   ANY    |      ANY      |
+-----+--------------+----------+----------+---------------+
Add VNFs by their Catalog SEQ (0 when done)
SEQ > 2
VNF included successfully!
SEQ > 3
VNF included successfully!
SEQ > 0

Add more Multi-SFC segments?
1. Yes
2. No
> 2

Source of the Multi-SFC network traffic at the first segment
1. Internal
2. External
> 2

+----------------+--------------------------------+
|       ID       |          Description           |
+----------------+--------------------------------+
|     arp_op     |          ARP opcode            |
|    arp_sha     |  ARP source hardware address  |
|    arp_spa     |    ARP source ipv4 address    |
|    arp_tha     |  ARP target hardware address  |
|    arp_tpa     |   ARP target ipv4 address     |
| dst_port_range |      Target port range         |
|    eth_dst     | Ethernet destination address  |
|    eth_src     |    Ethernet source address    |
|    eth_type    |      Ethernet frame type      |
|  icmpv4_code   |          ICMP code             |
|  icmpv4_type   |          ICMP type             |
|  icmpv6_code   |         ICMPv6 code            |
|  icmpv6_type   |         ICMPv6 type            |
| ip_dst_prefix  | IP destination address prefix |
|   ip_proto     |     IP protocol number         |
| ip_src_prefix  |   IP source address prefix    |
|   ipv6_dst     |   IPv6 destination address    |
|  ipv6_flabel   |      IPv6 Flow Label           |
|   ipv6_src     |    IPv6 source address         |
|  mpls_label    |         MPLS Label             |
|   vlan_id      |           VLAN ID              |
+----------------+--------------------------------+
```

```
Add the criteria by their respective ID (0 when done)
ID > ip_proto
Value > 6
ID > dst_port_range
Value > 80-80
ID > ip_dst_prefix
Value > 10.10.0.12/32
ID > 0

Define a name for this Multi-SFC instance (optional)
Name > example2

Multi-SFC example2 successfully instantiated!
```

APÊNDICE B – API DE TOLERÂNCIA A FALHAS DA RFT

Este apêndice apresenta a API de tolerância a falhas da RFT. A biblioteca da RFT disponibiliza um conjunto de operações para os desenvolvedores adicionarem serviços de tolerância a falhas aos xApps. A RFT pode ser vinculada aos xApps assim como outras bibliotecas do RIC, tais como RMR e SDL. A API da RFT consiste das seguintes operações:

- *init(callbacks)*: inicializa uma instância da RFT. Uma *thread* que implementa o componente *Dispatcher* (descrito na Seção 5.6) é criada para processar as mensagens endereçadas para a RFT e três funções de *callback* devem ser registradas pelo xApp por meio do parâmetro *callbacks*. As funções de *callback* são descritas mais a frente. A operação *init* também inicializa o subsistema de gerência do grupo de réplicas do xApp;

- *enqueue(msg)*: adiciona a mensagem *msg* em uma fila de tarefas (descrita na Seção 5.6) que a RFT deve executar em ordem. Apenas mensagens específicas e endereçadas para a RFT, reconhecidas por seus tipos, são enfileiradas pela réplica do xApp usando esta operação. Posteriormente, as mensagens são retiradas da fila pelo *Dispatcher* e distribuídas para os respectivos componentes da RFT também descritos na Seção 5.6;

- *get_role(relement)*: operação que permite que uma réplica do xApp identifique o seu papel em relação a um determinado elemento da RAN (*relement*), permitindo que cada réplica do xApp estabeleça o seu comportamento em uma situação específica. Três tipos de papéis podem ser retornados por esta operação: *primary*, *backup* ou *None*. O papel *primary* significa que a réplica do xApp é responsável por manter e atualizar os contextos daquele elemento da RAN e que, portanto, as informações dos contextos são as mais atualizadas. O papel *backup* denota que a réplica do xApp mantém apenas uma cópia dos contextos daquele elemento da RAN e que as informações dos mesmos podem não estar totalmente atualizadas. Uma réplica do xApp que tem o papel *backup* pode processar a mensagem de forma diferente do que uma réplica que tem o papel *primary*. O papel *None* significa que aquela réplica do xApp não possui qualquer informação de contextos sobre aquele elemento da RAN;

- *replicate(cmd, context, key, value, vlen)*: operação chamada pela réplica primária para executar em sequência e replicar um determinado comando da máquina de estado implementada no xApp. A operação recebe como parâmetros o comando (*cmd*) que deve ser executado na máquina de estado, o identificador do contexto (*context*), a chave que será alterada (*key*), o valor da respectiva chave (*value*) e o tamanho em bytes (*vlen*) transportado em (*value*). Cada comando da máquina de estado é representado por um *log entry* que é adicionado em sequência no componente *log* também descrito na Seção 5.4; No *backup*, a RFT faz a chamada da *callback apply* e executa o comando passado em *cmd* para atualizar as informações da cópia local do contexto;

- *(*apply)(cmd, context, key, value, vlen)*: função de *callback* registrada pela réplica do xApp para executar os comandos da sua máquina de estado permitindo atualizar as informações dos contextos. Os parâmetros desta função são passados para a máquina de estado no xApp e correspondem aos mesmos parâmetros recebidos na operação *replicate*. A função *apply* é chamada pela RFT em uma réplica primária sempre que um novo comando da máquina de estados do xApp é adicionado no *log* através da

operação *replicate*. No *backup*, a *callback apply* é chamada sempre que uma mensagem de replicação é recebida da réplica primária;

- *(*take_snapshot)(contexts, nctx, data)*: esta função de *callback* é chamada pela RFT para serializar a máquina de estado implementada em uma determinada réplica primária do xApp. Apenas os contextos que são mantidos e atualizados pela réplica primária são serializados. A RFT utiliza o parâmetro *contexts* para informar a lista de contextos primários que a máquina de estado do xApp deve serializar. O parâmetro *nctx* indica a quantidade de contextos primários contidos na lista *contexts*. O parâmetro *data* corresponde à região de memória em que o estado serializado será armazenado. É responsabilidade da máquina de estados do xApp alocar a região de memória para armazenar o *snapshot* em *data*. A *callback take_snapshot* ainda deve retornar o tamanho total do *snapshot*, em bytes, para a RFT. Um exemplo de serialização da máquina de estado de um xApp é dividir as variáveis de estado em grupos de bytes. Cada grupo de bytes pode consistir de "*um identificador de contexto e um par de chave-valor*" e ser representado da seguinte maneira: `clen|context|klen|key|vlen|value` em que `clen`, `klen` e `vlen` representam respectivamente o comprimento em bytes dos campos `context`, `key` e `value`. Além disso, a quantidade de grupos de bytes pode ser adicionada no cabeçalho do *snapshot* para indicar o número de grupos que foram serializados; e,

- *(*install_snapshot)(data)*: esta função de *callback* é chamada pela RFT em uma réplica *backup* para desserializar/instalar *snapshots* recebidos de uma réplica primária. A RFT disponibiliza o *snapshot* para a máquina de estado do xApp por meio do parâmetro *data*. Considerando o exemplo anterior da serialização, a máquina de estados do xApp inicialmente faz a leitura do cabeçalho do *snapshot* (*i.e.*, a quantidade de grupos) e, em seguida, desserializa cada grupo de bytes e atualiza as informações locais dos respectivos contextos.

APÊNDICE C – PUBLICAÇÕES

Este apêndice lista as publicações obtidas durante o desenvolvimento desta Tese de Doutorado.

1. **Alexandre Huff**, Matti Hiltunen, Elias P. Duarte Jr. (2021). RFT: Scalable and Fault-Tolerant Microservices for the O-RAN Control Plane. Em *2021 IFIP/IEEE International Symposium on Integrated Network Management*, Bordeaux, France. (Aceito para publicação)

2. **Alexandre Huff**, Giovanni Venâncio, Vinicius Fulber Garcia, Elias P. Duarte Jr. (2020). Building Multi-domain Service Function Chains Based on Multiple NFV Orchestrators. Em *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, páginas 19–24, Leganes - Madrid, Spain.

3. **Alexandre Huff**, Giovanni Venâncio, Leonardo da C. Marcuzzo, Vinicius Fulber Garcia, Carlos R. P. dos Santos, Elias P. Duarte Jr. (2018). A Holistic Approach to Define Service Chains Using Click-on-OSv on Different NFV Platforms. Em *2018 IEEE Global Communications Conference (GLOBECOM)*, páginas 1–6, Abu Dhabi, UAE.

4. **Alexandre Huff**, Giovanni Venâncio, Elias P. Duarte Jr. (2019). Multi-SFC: Orquestração de SFCs Distribuídas sobre Múltiplas Nuvens em Múltiplos Domínios com Múltiplas Plataformas NFV. Em *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, volume 37, páginas 777–790, Porto Alegre, RS, Brasil. SBC.

5. **Alexandre Huff**, Giovanni Venâncio, Leonardo da C. Marcuzzo, Vinicius Fulber Garcia, Carlos R. P. dos Santos, Elias P. Duarte Jr. (2018). Uma Abordagem Holística para a Definição de Service Chains Utilizando Click-on-OSv sobre Diferentes Plataformas NFV. Em *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, volume 36, Porto Alegre, RS, Brasil. SBC.

6. Vinicius Fulber Garcia, Elias P. Duarte Jr., **Alexandre Huff**, Carlos R. P. dos Santos (2020). Network Service Topology: Formalization, Taxonomy and the CUSTOM Specification Model. *Computer Networks*, 178:107337.

7. Vinicius Fulber Garcia, Leonardo da C. Marcuzzo, **Alexandre Huff**, Lucas Bondan, Jeferson C. Nobre, Alberto Schaeffer-Filho, Carlos R. P. dos Santos, Lisandro Z. Granville, Elias P. Duarte Jr. (2019). On the Design of a Flexible Architecture for Virtualized Network Function Platforms. Em *2019 IEEE Global Communications Conference (GLOBECOM)*, páginas 1–6, Waikoloa, HI, USA.